Relevant Artifacts and User Involvement in Scrum

Master Thesis

Nicklas Holm Jørgensen

Aalborg University
Department of Computer Science
Selma Lagerlöfs Vej 300
DK-9220 Aalborg Ø

AALBORG UNIVERSITY
STUDENT REPORT

Department of Computer Science
Selma Lagerlöfs Vej 300
DK-9220 Aalborg Ø
http://www.cs.aau.dk

Title:
Relevant Artifacts and User Involvement in Scrum

Topic:
Information Systems

Project Period:
Spring Semester 2017

Project Group:
is109f17

Participant(s):
Nicklas Holm Jørgensen

Supervisor(s):
Jan Stage

Copies: 1

Page Numbers: 37

Date of Completion:
June 07, 2017

Abstract:

This report presents empirical studies performed in collaboration with Department of Clinical Biochemistry at Aalborg University Hospital and includes two papers presenting the empirical studies. The first paper investigates which artifacts have perceived and practical relevance when developing with Scrum. We find that there are both differences and similarities in perceived and practical relevance of artifacts. Artifacts modeling the system was perceived to be less relevant but had practical relevance. Prototypes, work modeling, and a vision of the intended system had both perceived and practical relevance. The second paper investigates direct user involvement in Sprints, and what this is relevant for in relation to Design decisions and Design process—and to what extent direct user involvement influences the development process' contingencies. We find that direct user involvement was relevant to several things related to both Design decisions and Design process. Lastly, we identify complexity and uncertainty as contingencies in the development process; and that direct user involvement influence both.

Preface

This report presents a Master Thesis in Information Technology at Aalborg University. The Master Thesis is a requirement for the last semester of the Master Programme (MSc) in IT Design and Application Development.

The report consists of four chapters and two academic papers in CHI format available in the appendix. The report is organized as follows: Chapter 1 is an introduction presenting the problem statement and research questions; Chapter 2 summarizes the contributions made by the two academic papers; Chapter 3 describes the research method used in the two papers; Chapter 4 presents the conclusion, limitations, and future work.

During the entirety of the empirical studies presented in this report, we have been in close collaboration with the Department of Clinical Biochemistry at Aalborg University. Without the participation and patience of the staff, there would be no studies to present. We would, therefore, like to extend our gratitude to all participants from this department.

Finally, special thanks to Jan Stage for his guidance, feedback, and continued support.

Contents

Chapter 1

Introduction

Agile software development (ASD) represents different software development methods all subscribing to the same values of embracing change, being lightweight, and promoting flexibility in the development process [4]. Among the different methods characterized as being agile, Scrum is one of the most widely used in industry [9].

While Scrum, and ASD in general, promise to deliver useful software, it is also being criticized from different fronts. Part of this criticism is that agile methods disregard software engineering practices [10] by denouncing analysis and design typically associated with plan-driven approaches to software development. Another part of the criticism is that the end-user is neglected in the development process [1], meaning that agile methods may focus on developing useful but not explicitly *usable* software.

In light of this, practitioners are beginning to recognize the need for *some* Design Upfront and the importance of user involvement in the development process. However, Design Upfront and direct user involvement in Scrum is not without challenges either. Challenges that empirical studies need to investigate in search of potential solutions and valuable insights.

One such challenge is that empirical studies offer no consensus regarding which artifacts to include in upfront analysis and design—and typically only focus on the importance of artifacts pertaining to certain aspects of analysis and design. Thus, in practice, leaving out other possible aspects of analysis and design which could prove relevant during the development process.

Another is that there is lacking empirical studies investigating effective user involvement strategies, providing context-specific suggestions as to when and how user involvement is relevant during Sprints. Empirical studies point to problems of direct user involvement in Scrum stemming from limited resources [6], suggesting that a more beneficial user involvement strategy means involving users only when it is deemed relevant—and in a relevant way.

However, to arrive at the long-term goal of effective user involvement strategies in Scrum, we must first understand the influence of direct user involvement on contingency factors and investigate extent and outcome of direct user involvement [2].

1.1 Research questions

In response to the challenges posed by Design Upfront and direct user involvement in Scrum, we aim at addressing the research gap by presenting and investigating the following problem statement.

Problem statement: How can we help developers choose relevant upfront analysis and design artifacts, and understand how direct user involvement influence their Design decisions in Scrum?

To answer our problem statement, we present two research questions aimed at addressing Design Upfront and direct user involvement in Scrum.

Research question #1: Which upfront analysis and design artifacts have perceived and practical relevance when developing with Scrum?

The first research question address Design Upfront in Scrum by investigating which analysis and design artifacts have perceived and practical relevance. Thus, the hope is to arrive at insights on relevant analysis and design artifacts when developing with Scrum—and provide guidance to the extent of upfront analysis and design.

Research question #2: What is direct user involvement relevant for in relation to Design decisions and Design process—and to what extent does direct user involvement influence the development process' contingencies?

The second research question address direct user involvement in Scrum. Before achieving the long-term goal of effective user involvement strategies, we find that we must first understand how direct user involvement influence developers' decision-making and the development process. Thus, we set out to provide empirical insights on direct user involvement in relation to Design decisions, Design process, and influence on development process contingency factors.

1.2 Case

To investigate the problem statement and research questions presented above, we have been in close collaboration with Department of Clinical Biochemistry at Aalborg University Hospital. The department is responsible for servicing the entire hospital, which includes taking several hundred blood samples a day.

The bioanalysts working at Clinical Biochemistry are the ones taking the blood samples patients admitted to the hospital, and they carry around multiple different devices including multiple mobile phones, a PDA barcode scanner, and a pager. However, none of these devices have capabilities not found in a modern smartphone. We, therefore, decided to develop an Android smartphone application consolidating all these different devices. This would potentially make the daily work of the bioanalysts much easier and more convenient.

The close collaboration with the staff at Clinical Biochemistry provides a valuable opportunity in relation to the presented problem statement and research questions. Not only does the collaboration provide an opportunity to develop a useful and beneficial solution for the bioanalysts, but also the opportunity of investigating the relevance of analysis and design artifacts seen from a developer perspective during development. Further, it is a unique opportunity to investigate what user involvement is relevant for in relation developers' Design decisions and Design process.

Chapter 2

Contributions

This chapter presents the two papers that represent the main parts to this master thesis. Each of the two papers will be presented with a summary and main findings. A chronological reading of the two papers is recommended; however, they investigate different aspects of the proposed problem statement so they can be read individually.

2.1 Contribution 1

Nicklas Holm Jørgensen. The relevance of upfront analysis and design artifacts in scrum. *Department of Computer Science, Aalborg University*. Aalborg, 2017.

The first contribution presents an empirical study investigating the perceived and practical relevance of analysis and design artifacts when developing with Scrum. The goal of the study was to investigate the relevance of different artifacts informing both system-focused and user-focused considerations in Scum, and further arrive at some practical recommendations to analysis and design artifacts.

The empirical study consists of two different elements: An empirical survey conducted with agile practitioners, and an exploratory study conducted while developing an Android application. The first element presents findings from a survey conducted with 13 agile practitioners from seven different companies. The goal of the survey was to investigate the practitioners' perceived usefulness of selected analysis and design artifacts. The second element presents findings from a development diary documenting how many times the selected analysis and design artifacts were used in practice, and for what purpose.

We find that there are both differences and similarities of the perceived and practical relevance of selected analysis and design artifacts. Artifacts primarily modeling the system were perceived to be less relevant in the survey, but had practical relevance during the development of the application. However, prototypes, work modeling, and a vision of the intended system had both perceived and practical relevance. In closing, we propose to choose artifacts systematically, value simplicity of artifacts and address possible aspects of analysis and design.

2.2 Contribution 2

Nicklas Holm Jørgensen. An exploratory study of direct user involvement and design decisions in scrum sprints. *Department of Computer Science, Aalborg University*. Aalborg, 2017.

The second contribution presents an exploratory study investigating what direct user involvement is relevant for in relation to Design decisions and Design process—and to what extent direct user involvement influence the development process' contingencies. The goal of the study was to better understand how user involvement in Scrum Sprints influence developers' decision-making and problem-solving.

The exploratory study presents finding from a diary kept while developing an Android application for supporting the work practice of bioanalysts at Department of Clinical Biochemistry at Aalborg University Hospital. During a development period spanning three months in total, we kept a diary documenting Design decisions, Design process and how direct user involvement in Sprints influenced the development process.

We find that direct user involvement in Sprints was relevant for several things concerning Design decisions. Direct user involvement helped in defining the satisficing behavior of the developer exhibited throughout the development process. It also helped in adjusting aspiration levels to align with the users' and was very relevant to the transfer of tacit knowledge from user to developer. Further, direct user involvement was relevant for generation and validation of requirements during the Design process.

Lastly, we find that uncertainty and complexity were contingency factors throughout the development process; and that direct user involvement influenced both. Uncertainty was continuously generated during Sprints, and direct user involvement helped reduce uncertainties by evaluation of prototypes and through social interaction. However, user involvement also introduced complexity that requires appropriate responses of the developer.

Chapter 3

Research method

In this chapter, we will present and discuss the research method and research techniques used in the empirical studies. We will first present the research method, and discuss strengths and weaknesses of this method. Then we will proceed to present the research techniques used concerning the presented research questions, and further, discuss their strengths and weaknesses.

3.1 Case study

The two research questions presented in this summary report were both addressed using a case study—however, the research techniques and purpose differed (see Table 3.1 for an overview).

Research question	Method	Technique	Purpose
RQ1	Case study	Survey Diary	Exploration Explanation
RQ2	Case study	Diary	Exploration Description

Table 3.1: Research method and research techniques used in relation to research questions.

The first research question was addressed with a case study used for exploration and explanation (as described in [7]). One of the main characteristics of an exploration case study is that the researcher starts with an *"an incomplete or preliminary understanding of a problem and its context"* [7]. In our case, we began with a preliminary understanding of the problems associated with up-front analysis and design in ASD. We did not, however, have any predefined solutions to said problems. The purpose of the case study is the exploration of possible solutions and their merit. Further, based on our empirical study we sought to provide an understanding of the perceived and actual relevance of different analysis and design artifacts when developing with Scrum—in an attempt to arrive at an explanation for the various uses of these artifacts in an agile context.

The second research question was addressed with a case study used for exploration and description. In this case, we started with a preliminary understanding of the problems associated with direct user involvement in Scrum—and explored what direct user involvement is relevant for in relation to Design decisions and Design process. And so, the case study is used to describe how users influence developers' decision-making and problem-solving during the development process, and the lessons learned. Even though the findings in this study were depended on the specific case and context, the purpose of the descriptive case study is also to present lessons learned that could provide insights in other cases [7].

Strengths and weaknesses

As expressed in [11], the primary strengths of case studies are rich descriptions and explanatory evidence. Further, case studies can provide insights into phenomena otherwise difficult to investigate [7]. However, case studies also present some challenges. The most common drawbacks being the typically high cost of performing case studies, the time-consuming nature of case studies, and the limited generalizability of findings [11]. See an overview of strengths, weaknesses, and countermeasures in Table 3.2.

Strengths	**Weaknesses**	**Countermeasures**
Rich data	High cost	Multiple data sources
Valuable insights	Time-consuming	Direct access
Explanatory evidence	Limited generalizability	No generalization

Table 3.2: Strengths and weaknesses of used research techniques.

The high cost and time-consuming nature of performing case studies were not a deciding factor in our case since the empirical studies were confined by practicalities related to deadlines. Further, we had direct access to participants from the collaborating hospital; providing access when needed, and to the extent needed. As a countermeasure to the limited generalizability, we used in one of the empirical studies multiple research techniques to provide contrasting views. Further, we targeted specific respondents with different backgrounds in the survey as to negate sampling bias. Lastly, as a countermeasure we seek not to generalize from the empirical studies; but rather provide insights that might prove useful and interesting for other cases.

3.2 Surveys

Strengths	**Weaknesses**	**Countermeasures**
Easy to collect data	Biased data	Not told to recollect
No interviewer effect	No probing	Open-ended questions
Convenient	Shallow data	Targeted population
	Sampling bias	

Table 3.3: Strengths and weaknesses of using surveys.

Surveys are very common as research techniques and data collection methods across all fields of research. The main strengths of surveys are that they are very easy to administer, and consequently very useful for collecting many responses quickly. Further, surveys are typically very convenient for respondents compared to e.g. interviews—and they do not suffer from interviewer effects influencing responses [3]. Surveys do, however, also have weaknesses. Firstly, if surveys are used concerning patterns of usage, there is a high risk of biased data as a consequence of the difficulty of recollect, or because of wrong estimates [7]. Secondly, surveys typically produce shallow data since the researcher have no possibility of probing the respondents [3]. Lastly, surveys suffer from sampling bias as a result of surveying the wrong population [7].

In an attempt to mitigate the weaknesses of surveys, we have introduced different countermeasures in the empirical study. Respondents are not surveyed about patterns of usage but rather asked to give their evaluation of perceived usefulness. Further, respondents were targeted specifically in relation to the research questions; thus only surveying respondents who were familiar with agile development represented by Scrum. Finally, as a countermeasure to shallow data, respondents were provided with the opportunity of expanding on their evaluations in free text. Thus providing qualitative data complementing the quantitative data also collected via the survey. An overview of strengths, weaknesses, and countermeasures can be seen in Table 3.3.

3.3 Diaries

Strengths	**Weaknesses**	**Countermeasures**
Rich data	Time-consuming	Defined structure
Reliable	Process of attrition	Limited time-period
Good for sequencing	Failure to recall	Clear purpose
		Electronic format

Table 3.4: Strengths and weaknesses of using diaries.

Diaries as a research technique and data collection method are commonly used in sociology and history [7], however, diaries are not uncommon in systems development either [8]. The main strengths of diaries are that they tend to provide more reliable data when estimates of both time and frequency are required [3]. And more, that they are a suitable data collection method when observational methods or surveys are either insufficient or impractical [7]. In the case of describing e.g. why certain solutions were preferred over others, diaries provide much more insight and rich data than do experimental or observational data collection methods. However, there are also weaknesses related to using diaries as data collection. One being that diaries typically are very time-consuming, and so result in a "process of attrition" [3]. Further, there is the chance that the diary suffers from biased data if the writer does not record details promptly—thus forgetting important details or the decision process.

As countermeasures to the weaknesses of using diaries, we have implemented practical advice for using this data collection method as suggested by [5]. This includes a defined structure as not to forget important details and to state the purpose of the diary. Further, the diary was written during a predefined period to mitigate the process of attrition. Lastly, to countermeasure the time-consuming nature of diaries, all entrances were written directly (without a draft) and in electronic format.

Chapter 4

Conclusion

This chapter presents conclusions to the two research questions, and the problem statement presented in this summary. We first summarize conclusions of the research questions, which leads us to the conclusion of the overall problem statement. We then present limitations of our studies, and, finally, propose future work.

4.1 Research questions

Research question #1: Which upfront analysis and design artifacts have perceived and practical relevance when developing with Scrum?

We find that there was both differences and similarities in perceived and practical relevance of selected analysis and design artifacts.

Regarding differences in perceived and practical relevance, especially artifacts modeling the system was perceived to be less relevant. However, we find that these artifacts had high practical relevance. Contrary to this, we find that prototypes, work modeling, and a vision of the intended system had both perceived and practical relevance. Further, we find that artifacts were used in two different ways during development: Directly and tacitly.

Lastly, we propose practical implications. We suggest to choose artifacts systematically, value simplicity when developing artifacts, and to address possible aspects of analysis and design.

Research question #2: What is direct user involvement relevant for in relation to Design decisions and Design process—and to what extent does direct user involvement influence the development process' contingencies?

We find that direct user involvement was relevant for several things in relation to Design decisions and Design process—and that direct user involvement influenced identified contingencies in the development process.

Specifically, direct user involvement was relevant for defining the satisficing behavior of the developer; was valuable in adjusting aspiration levels, and was relevant to the transfer of tacit knowledge from user to developer. Further, we find that direct user involvement was relevant in generating and validating requirements during the Design process.

Lastly, we identify complexity and uncertainty as contingencies during the development process—and that direct user involvement had an influence on both. Uncertainty was produced throughout the development process, and user involvement helped reduce uncertainties. In turn, direct user involvement introduced complexity requiring appropriate responses from the developer.

Problem statement: How can we help developers choose relevant upfront analysis and design artifacts, and understand how direct user involvement influence their Design decisions in Scrum?

We investigated perceived and practical relevance of analysis and design artifacts in Scrum—and provided empirical insights on differences and similarities. Further, we propose practical implications related to developing analysis and design artifacts. Thus, our empirical study contributes with insights that could prove useful for practitioners when developing and deciding on analysis and design artifacts in Scrum.

Second, we provide empirical evidence on direct user involvement and influence on Design decisions and Design process. Building on literature on user involvement, we contribute with insights that could prove valuable in future empirical studies on effective user involvement strategies in Scrum. Further, we identify contingency factors during development and reveal findings on how these are influenced by direct user involvement during Sprints. This empirical evidence also contributes to our understanding of direct user involvement and influence on Design decisions.

4.2 Limitations

We acknowledge the limitations of our work, and especially the nature of case studies. Both empirical studies reveal findings from a case study, and we are aware that this has could have an impact on the conclusions presented in this summary. Had another case been chosen, and had the case study been conducted under different circumstances, findings could prove different than what we have presented in our empirical studies. However, we have tried to negate this by including multiple research techniques—and seek not generalize based on our results, but provide valuable insights.

We also acknowledge that our research and findings are conducted and interpreted by ourselves. Thus, it presents the issue of researcher bias—and whether findings are interpreted objectively. It is clear that our view of findings could differ from other researchers. However, in an attempt to negate this, we have used multiple research techniques including both qualitative and quantitative research techniques. Further, we have aimed at providing clarity and transparency in our data analysis.

4.3 Future work

Additional empirical studies should be performed to expand on our findings. Specifically, it could provide valuable insights to investigate the relevance of different analysis and design artifacts in larger development teams. And even more so, investigate how different development situations affect the relevance of analysis and design artifacts in Scrum. Thus, arriving at empirical evidence to further help developers choose which analysis and design artifacts to develop depending on e.g. development situation.

Our findings provide insights on direct user involvement in Scrum Sprints. However, further empirical studies should build on our empirical evidence. Studies investigating the influence of direct user involvement in larger teams could add further insights not revealed in our exploratory study. Also, empirical studies investigating direct user involvement in Sprints over longer periods of time could provide valuable insights for developers. Specifically, it could help in understanding how users' needs change during the development process; and how this affects developers' decision-making.

Lastly, there is lacking empirical studies investigating contingencies during the development process—and how these influence effective user involvement strategies. Our findings identify complexity and uncertainty as contingencies and that direct user involvement influence both. Empirical studies could build on our research by suggesting how these contingency factors could be used to decide on effective strategies for user involvement in Sprints.

Bibliography

[1] Stefan Blomkvist. Towards a model for bridging agile development and user-centered design. *Human-centered software engineering—integrating usability in the software development lifecycle*, pages 219–244, 2005.

[2] Manuel Brhel, Hendrik Meth, Alexander Maedche, and Karl Werder. Exploring principles of user-centered agile software development: A literature review. *Information and Software Technology*, 61:163–181, 2015.

[3] Alan Bryman. *Social Research Methods*. Oxford University Press, 2008.

[4] Jim Highsmith and Alistair Cockburn. Agile software development: The business of innovation. *Computer*, 34(9):120–127, 2001.

[5] Leif Obel Jepsen, Lars Mathiassen, and Peter Axel Nielsen. Back to thinking mode: diaries for the management of information systems development projects. *Behaviour & Information Technology*, 8(3):207–217, 1989.

[6] Kati Kuusinen, Tommi Mikkonen, and Santtu Pakarinen. Agile User Experience Development in a Large Software Organization: Good Expertise but Limited Impact. In Marco Winckler, Peter Forbrig, and Regina Bernhaupt, editors, *Human-Centered Software Engineering*, volume 7623. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[7] Jonathan Lazar, Jinjuan Heidi Feng, and Harry Hochheiser. *Research methods in human-computer interaction*. John Wiley & Sons, 2010.

[8] Peter Naur. An experiment on program development. *BIT Numerical Mathematics*, 12(3):347–365, 1972.

[9] Tina Øvad and Lars Bo Larsen. The prevalence of ux design in agile development processes in industry. In *Agile Conference (AGILE), 2015*, pages 40–49. IEEE, 2015.

[10] Steven Rakitin. Manifesto elicits cynicism. *IEEE Computer*, 34(12):4, 2001.

[11] Judy L. Wynekoop and Sue A. Conger. *A review of computer aided software engineering research methods*. Department of Statistics and Computer Information Systems, School of Business and Public Administration, Bernard M. Baruch College of the City University of New York, 1992.

Appendix A

Appendix

(1) Nicklas Holm Jørgensen. The relevance of upfront analysis and design artifacts in scrum. *Department of Computer Science, Aalborg University*. Aalborg, 2017.

(2) Nicklas Holm Jørgensen. An exploratory study of direct user involvement and design decisions in scrum sprints. *Department of Computer Science, Aalborg University*. Aalborg, 2017.

The Relevance of Upfront Analysis and Design Artifacts in Scrum

Nicklas Holm Joergensen
Aalborg University
Department of Computer Science
Aalborg, Denmark
nhjo10@student.aau.dk

ABSTRACT
Agile software development denounces the Big Design Upfront typically associated with plan-driven approaches to software development, which has led to criticisms of encouraging hacker mentality and not delivering usable products to the customer. Recognizing the need for some Design Upfront, agile practitioners point to the relevance of different upfront analysis and design artifacts addressing mainly system-focused or user-focused considerations. This paper presents an empirical study of the perceived and practical relevance of upfront analysis and design artifacts when developing with Scrum. The study consists of two different elements: An empirical survey with agile practitioners, and an exploratory study conducted while developing an Android application. We find that system-focused artifacts were perceived as less relevant by survey respondents but had practical relevance. Further, we find that prototypes, work modeling, and a vision of the intended system had both perceived and practical relevance. In closing, we provide insights into some practical implications.

Author Keywords
User-Centered Design; Scrum; agile development; upfront analysis and design

INTRODUCTION

Agile approaches to software development represent a departure from traditional, plan-driven approaches with emphasis on extensive planning, analysis, and documentation. Agile software development (ASD) encapsulates many different methods all emphasizing the same fundamental values described in the Agile Manifesto [15]. ASD comes as a reaction to the notion of optimal and predictable solutions available for every problem, and focus instead on incremental delivery and minimal documentation [12]. Proposing effective ways of managing changing business—and customer requirements—agile approaches have in recent years gained momentum in industry [29] with Scrum being one of the most widely used in industry [38].

Denouncing the Big Design Upfront (BDUF) associated with plan-driven approaches to software development, ASD is, however, eliciting skepticism from practitioners favoring plan-driven approaches. A typical example being Rakitin [28] commenting that ASD is encouraging hacker mentality and disregarding predictable software engineering practices. Thus resulting in a product not reflecting what the customer expects, and delivered not within the agreed upon timeframe. Viewing plan-driven and agile approaches as different ends of the same spectrum, it is suggested that the best result comes from balancing the two [7]. Modern companies require rapid value from usable and reliable software, suggesting the limitation of adhering strictly to only a plan-driven or agile approach.

Recognizing the value of *some* Design Upfront, agile practitioners point to developing different artifacts as part of upfront analysis and design. Some practitioners focus on relevant artifacts related to system-focused considerations in ASD. This include creating a metaphor of the intended system [5], or defining a broad starting point architecture [14]. Further examples point to the value of using modeling tools, such as UML, when engaged in upfront analysis and design [1, 3, 36]. Other practitioners focus primarily on integrating User-Centered Design (UCD) practices in ASD, criticizing agile approaches for not delivering usable products [11]. In this regard, upfront analysis and design is typically concerned with developing user-focused artifacts such as prototypes, scenarios and personas [9].

Relevant literature does not, however, point to any consensus regarding which artifacts to include in upfront analysis and design. Further, empirical studies typically focus only on the importance or relevance of including either system-focused or user-focused artifacts in ASD; thereby leaving out other possible aspects of analysis and design. Addressing criticisms regarding both system-focused and user-focused challenges in ASD, upfront analysis and design in ASD need to accommodate several aspects of software development.

This paper presents an empirical study of the perceived and practical relevance of upfront analysis and design artifacts when developing with Scrum. The study consists of two complementing elements. The first element is an empirical survey conducted with agile practitioners. The practitioners were asked to evaluate the perceived usefulness of selected analysis and design artifacts when developing an IT system with Scrum. The second element is an exploratory study of the practical use of selected analysis and design artifacts. While developing an Android application for supporting the daily work of bioanalysts from the local university hospital, we kept diary documenting the practical relevance of analysis and design artifacts.

The rest of this paper is organized as follows: In the following section, we summarize related work on analysis and design in ASD. We then present a literature survey aimed at identifying artifacts to be evaluated in the empirical study. Then we present the empirical study and relevant findings. Finally, we provide a discussion and conclusion of this study.

RELATED WORK

In the following, we will present an overview of relevant literature regarding upfront analysis and design in ASD. We have identified empirical studies focusing on either one of two categories: System-focused design upfront, or user-focused design upfront.

System-focused design upfront

There are few empirical studies regarding system-focused design upfront in ASD, and practitioners' perceived relevance. Falessi et al. [13] did a survey with 72 experienced developers from IBM in Rome, investigating the relevant use of software architecture in an agile context. The study report that respondents perceive the most relevant use of software architecture to be in relation to communication, and as input to subsequent system design and development activities. The study does not, however, investigate the perceived or practical relevance of other system-focused artifacts than software architecture.

Other empirical studies focus on activities and practices with regards to developing system-focused design upfront in ASD. Babar [4] did an exploratory study of an agile software company provisioning financial services; highlighting architecture related practices and challenges in ASD. The study report the use of initial analysis and design phases when developing with Scrum. Participants in the study report the frequent use of a project wiki describing design decisions and component diagrams, in combination with design meetings, when communicating architectural design. The study does not, however, report on the perceived or practical relevance of other system-focused or user-focused artifacts.

Prause and Durdik [27] conducted interviews with 37 software engineering experts from industry and academia, investigating architectural design and documentation in ASD. The study report a majority of respondents believing explicit architectural design to be beneficial. Further, that the majority of respondents engaged in architectural activities in ASD perform some actual initial analysis and design. The study does not, however, provide any guidance as to what these activities may be—or report on how respondents use system-focused artifacts in practice.

Finally, Yang et al. [37] performed a systematic mapping study, covering literature on the combination of architecture and ASD. Among the various architectural activities identified in industry, an architectural description was most commonly identified with other activities receiving limited attention. A supposed reason for this being the significant effort and time related to other architectural activities, thus reducing the relevance of including this type of initial analysis and design in agile approaches. Though the study highlights different activities related to developing an architecture in ASD, it does not report on the perceived or practical relevance of other system-focused artifacts.

In summary, previous empirical studies investigate the perceived relevant use of software architecture, and the practical relevance of some architecture related analysis and design artifacts. However, none of the empirical studies focus on the practical relevance of different system-focused artifacts. There are reports of the perceived relevance or value in system-focused considerations, but the empirical studies does not investigate the practical relevance—and how these artifacts are used during development. Neither do empirical studies provide any guidance as to which system-focused artifacts to develop during upfront analysis and design. Lastly, previous empirical studies do not address user-focused concerns in ASD.

User-focused design upfront

There are, as with system-focused design upfront, few empirical studies on user-focused design upfront in ASD and practitioners' perceived value or relevance. Hussain et al. [18] did a survey on User-Centered Design (UCD) in ASD conducted with 92 developers and usability professionals. The study identify the most common user-focused practices as lo-fi prototyping, conceptual designs, and observation studies of users. The majority of respondents perceive that integrating UCD activities in ASD has added value to the process and their teams, and that it has resulted in improvement of usability and quality of the final product. However, the study does not provide any guidance as to the practical relevance or use of user-focused artifacts.

Jia et al. [21] did an empirical study investigating practitioners' perceived usefulness of usability techniques, and how often they use them, in Scrum projects. The study was conducted with 35 IT professionals using both Scrum and various usability techniques in their development process. The study identify that the most commonly used usability techniques by respondents include workshops, lo-fi prototyping, and interviews. However regarding usefulness, respondents perceived formal usability evaluations, digital prototypes, and field studies as the most useful techniques. A limitation to this study is that it only investigates the perceived usefulness of usability techniques, and not the practical relevance when developing with Scrum.

Several other empirical studies has been conducted on UCD and usability in software development; mainly focusing on perceived value of usability work, and which practices are most common in industry. Vredenburg et al. [34] did an empirical survey with 103 UCD practitioners, investigating the overall impact of UCD activities and common UCD methods in industry. They conclude that UCD methods are generally perceived by practitioners to have improved the final product, but that there are no common standards for measuring actual effectiveness of usability work. The study further identify the most commonly used UCD methods in practice to be iterative design, usability evaluation, and task analysis.

Gulliksen et al. [17] performed a survey with 194 UCD practitioners in Sweden, investigating their work practices. The study reports practitioners' perceived value of different UCD methods and techniques used in practice; with lo-fi prototyping and interviews being among the highest rated, and personas and questionnaries being among the lowest. The respondents in this survey come from companies with different software development processes, however only a small percentage deem their process agile.

Lastly, Ji and Yun [20] did a survey with 184 IT development and 90 UCD practitioners from different industries and companies in Korea, investigating UCD and usability adoption and activities. They find that both developers and UCD practitioners value usability work, however UCD practitioners generally ascribe more value to usability work. Further, the study identify the most common usability methods as perceived by both developers and UCD practitioners with task analysis and the evaluation of existing systems as being the most common.

Summarizing, empirical studies on user-focused analysis and design investigate the most common activities and techniques in practice. Further, they report on practitioners' perceived relevance and benefits to user-focused analysis and design. Few studies, however, investigate the perceived or practical relevance of user-focused analysis and design artifacts in an agile context. Neither does empirical studies provide any consensus as to which artifacts to develop during upfront analysis and design in ASD. Finally, empirical studies on user-focused design upfront do not take into consideration the perceived or practical relevance of system-focused artifacts.

LITERATURE SURVEY

Following related work, we observe that literature on system-focused and user-focused analysis and design in ASD provide some proposals for potentially relevant activities and artifacts in relation to design upfront—but provide no consensus or systematic selection of artifacts. To identify common artifacts in industry to be evaluated in this empirical study, we have searched part of the relevant literature.

A preliminary literature survey on Google Scholar was conducted, with search criteria consisting of combinations of "agile and UCD integration," "agile user-centered design," "upfront analysis and design in agile," "agile modeling," and "architecture and agile." Results of this search were scanned for relevance by reading title and abstract. After the preliminary search, several papers concerned with developing upfront system-focused and user-focused artifacts were selected. These papers were further supplemented with other papers identified and referenced in recent literature reviews regarding UCD in ASD [9, 30].

Based on the literature survey, we identified common upfront analysis and design artifacts used in practice. However, since many of the papers identified were anecdotal evidence or experience reports, artifacts concerning the same purpose are often referred to using different terms. One example being work models that are referred to as both task flow [31], user flow [10], and work flow [6]. Hence, after identifying common artifacts, we abstracted terms seemingly concerning the same purpose as to gain an overview of which artifacts are most typical in industry. The result of the literature survey and abstraction of common artifacts can be seen in Table 1 on the following page.

Following our literature survey and abstraction of analysis and design artifacts, we compared our results to recent literature reviews on integrating UCD practices in ASD [30, 9]. Our results are consistent with both reviews, highlighting prototypes, personas, and scenarios as some of the most common practices and artifacts. We have not identified any literature review focusing on system-focused upfront design artifacts in ASD. However, our results are consistent with a recent systematic mapping study of architectural practices in ASD highlighting an architectural description as the most common architectural activity; e.g. developing an overall system vision or architectural modeling [37].

Categories of analysis and design artifacts

To cover possible aspects of upfront analysis and design, this paper introduces analytic categories representing different parts of a software development process. These categories are presented in Mathiassen et al. [24], and can according to the authors be considered a complete method for the analysis and design of IT systems.

The purpose of these categories is to introduce concepts that can be used to ensure, that selected artifacts cover possible aspects of upfront analysis and design. The analytic categories are defined as follows:

High-level Description An overall vision expressed in natural language, making it possible for developers, customers, and users alike to gain an understanding of the intended system.

Problem Domain Viewing the system's context from two complementary perspectives, the problem domain is the part of the context being administered, monitored, or controlled.

Application Domain Closely related to the problem domain, the application domain represents the part of the context that administrates, monitors, or controls.

System Architecture Structuring of main components and specification of their relationship on an abstract level.

Component Design Specification of the system's components on a lower level; e.g. interface component design or data model.

Each category is characterized by being concerned with a different aspect of analysis and design, and each informed by different analysis and design artifacts. The intention being that these categories will prove useful after identifying artifacts used in practice since it allows for explicit characterization of each artifact. Thus, providing an overview of which artifacts

Artifacts	Papers	Total
Prototypes	[36], [32], [35], [31], [33], [10], [6] [22], [8], [16]	10
Vision	[32], [31], [6], [33], [14]	5
Personas	[32], [26], [31], [6]	4
Scenarios	[32], [26], [2], [36]	4
Work models	[32], [31], [10], [6]	4
Data models	[3], [36], [14], [1]	4
Affinity diagrams	[6]	1
Storyboards	[6]	1
User Environment Design	[6]	1

Table 1. Abstraction of artifacts identified in literature survey (n = 15).

High-level Description	Problem Domain	Application Domain	System Architecture	Component Design
System definition	Event table	Personas	Component architecture	Prototypes
Affinity diagram	Statechart diagram	Scenarios		Model component
	Class diagram	Work modelling		

Table 2. Selected artifacts related to analytic categories.

inform which categories will help ensure that possible aspects of analysis and design are addressed.

Finally, common upfront analysis and design artifacts identified in the literature survey were selected and related to the analytic categories to ensure that possible aspects of analysis and design are covered by the selected artifacts. The result can be seen in Table 2.

Case

Having identified and selected artifacts to be evaluated, we performed an initial analysis and design phase. This was done on the basis of the case described in the following.

We have collaborated with Department of Clinical Biochemistry at Aalborg University Hospital during the empirical study presented in this paper. The department is responsible for servicing the entire hospital; which among other things include taking several hundred daily blood samples. The staff at the department collect and analyze requested blood samples, and consequently informs the requester of available results. The staff at the department carry around multiple different devices used in their daily work practice; e.g. different telephones, PDA scanner, and a pager. However, none of the devices have capabilities not found in a modern smartphone. Thus we imagined the task ahead was to consolidate the different devices into one, making the work of the bioanalysts at the department easier and more convenient.

Using this as our starting point, we conducted multiple observations and interviews at the hospital that provided data and material for developing the selected artifacts to be evaluated. This analysis and design phase lasted three weeks in total and was performed in December 2016.

EMPIRICAL STUDY

This paper aims at providing empirical evidence as to the perceived and practical relevance of analysis and design artifacts, and further provide guidance as to the extent of upfront analysis and design in ASD. We do this by conducting an empirical study consisting of two different elements.

1. The first element is findings from an empirical survey conducted with agile practitioners, investigating their perceived usefulness of the selected analysis and design artifacts.

2. The second element is findings from an exploratory study of the practical use of selected analysis and design artifacts.

In the following, we will first present the research method of the empirical survey conducted with practitioners, and then we will present the exploratory study. Finally, we present the findings from the two elements.

Element 1: Perceived relevance

The first element consists of an online survey with both close-ended multiple choice and open-ended questions. The survey was made up of 39 questions covering the different artifacts and demographic questions. As an introduction to the survey, participants were presented with the context of the different artifacts; namely the case presented previously. For each selected artifact presented in this paper, participants were shown a graphic depiction along with questions regarding that artifact. Participants were asked to evaluate their familiarity with each artifact on a 5 point Likert scale; ranging from *not familiar* to *very familiar*. Further, they were asked to evaluate how useful each artifact would be on a scale of 1-5, were they to begin developing a new IT system using Scrum. Finally, participants were asked to explicate their evaluation of each artifact's perceived usefulness using free text. However, this was not mandatory, so not all respondents answered this question.

The survey was aimed at Danish practitioners working with Scrum. The reason for this being that Scrum is one of the most widely used agile methods in practice [23], and so the survey used Scrum as the representative of agile methods. The survey was distributed through academic and personal networks, and 13 participants from seven different companies responded in total. Table 3 on the next page shows the participant profiles.

Participants responded to the survey on two separate occasions. Participants 1-7 responded from December 2016 to January 2017, and participants 8-13 responded from March

Participant	Background	Scrum role	Company
P1	UX designer	Team member	Large
P2	UX coach	Other	Large
P3	Developer	Team member	Medium
P4	UX designer	Team member	Medium
P5	Test manager	Team member	Large
P6	Project manager	Scrum master	Medium
P7	Developer	Team member	Large
P8	UX designer & developer	Other	Medium
P9	UX designer	Product owner	Large
P10	Development manager	Product owner	Medium
P11	UX designer	Product owner	Medium
P12	UX designer	Product owner	Large
P13	Team coach	Other	Large

Table 3. Overview of participants.

to April 2017. The only difference between the design of the two surveys was that the latter group of participants had the opportunity to optionally recommend further artifacts useful in upfront analysis and design as part of the survey. However, the first group of respondents was contacted shortly after their responses and was offered the opportunity to recommend further artifacts via phone or email.

All survey responses were given in Danish and have been translated verbatim into English for this paper.

Element 2: Practical relevance
The second element was performed as an exploratory study investigating the practical use of the selected analysis and design artifacts. We collaborated with Aalborg University Hospital while developing an Android smartphone application prototype to support the daily work practice of bioanalysts at the hospital. The application was developed using the selected upfront analysis and design artifacts as a starting point.

The study was further conducted with two volunteering staff from Department of Clinical Biochemistry at Aalborg University Hospital. The two participants were chosen to represent the role of Product Owner, thus being responsible for prioritization of User Stories on the Product Backlog. The first participant is working as a biochemist at the hospital, responsible for quality assurance and research; the second is working as a laboratory professional in close contact with the bioanalysts at the hospital.

During the development period spanning three months in total, we kept a development diary reflecting on the development process; noting how often the selected upfront analysis and design artifacts were used, and for what purpose. New entrances to the diary were written shortly after finishing each day of developing the application, so as not to forget details or relevant information. Further, the diary was written with a clear purpose in mind, and was written according to a predefined structure as recommended by [19]. Thus, the diary provides empirical evidence as to which artifacts had practical relevance for the development of the application.

The diary was originally written in Danish and all passages shown in this paper has been translated verbatim into English.

ELEMENT 1: FINDINGS
Key results from the survey investigating practitioners' perceived relevance of selected artifacts covering possible aspects of analysis and design are presented in the following. The respondents' average assessment of perceived usefulness and familiarity with the different artifacts can be seen in Table 4.

		U	F
High-level Description	System definition	3.54	3.62
	Affinity diagram	3.85	3.23
Problem Domain	Class diagram	2.38	3.15
	Event table	2.08	2.15
	Statechart diagram	2.46	2.85
Application Domain	Personas	2.85	3.77
	Scenarios	3.08	3.08
	Work modelling	4.31	3.54
System Architecture	Component architecture	2.85	2.46
Component Design	Prototypes	4.46	4.46
	Model component	2.46	2.77

Table 4. Average assessment of usefulness (U) and familiarity (F) on a scale of 1-5.

High-level Description
Respondents perceive the artifacts informing the high-level description as being useful. Viewing the results from Table 4, the analysis and design artifacts evaluated by practitioners are both rated well above the median value; giving the system definition a score of 3.54 and the affinity diagram a score of 3.85. Further, practitioners are overall familiar with both artifacts.

There are, however, a few nuances in the open-ended responses. One of these being that a vision of the intended system can be expressed in other ways than a prosaic system definition, and that this might be better than written, natural language. P5 expresses that "*It is still just prose and I would like some more specs*" and in the same vein, P4 says that "*[...] there are better ways of describing a solution to a professional than prose*". However, P4 also concurs that a prosaic system definition would be easy for customers to understand. P12 expresses that a system definition specifies the context of the intended solution, but "*[...] the prose in written form can be replaced by dialogue and communication about the solution*".

All respondents except for one perceive affinity diagrams as being a useful upfront analysis and design artifact. P12 says there is value in specifying information but that "*I would rather use for example a goal hierarchy*". P1 says "*[an] affinity diagram is a super tool to analyze large amounts of qualitative data*", and P2-P6 concur saying that it is useful for organizing and structuring data. Specifically, P2 express that "*affinity diagrams are the result of the observations I've done after field studies or interviews. It provides me with data and not perceptions*".

Summarizing the respondents' perceived usefulness of the artifacts informing the high-level description of the system, they ascribe value to both the system definition and the affinity diagram. One recurring theme is the possibility of developing a vision for the intended system in other ways that prose.

Problem Domain

The respondents are divided regarding their perceived usefulness of analysis and design artifacts informing the problem domain, but the tendency is that practitioners perceive them as less useful before the first Sprint. Thus rating all analysis and design artifacts under the median value; giving the class diagram an average of 2.38 out of 5, the event table an average rating of 2.08 out of 5, and the statechart diagram an average rating of 2.46 out of 5.

There could be multiple reasons for this, but one suspected reason is the difference in the respondents' job function and familiarity with the different analysis and design artifacts. P1 gives the different artifacts the lowest possible rating with the reason that they are unfamiliar. The same tendency can be seen in the responses from P2, P4, P8 and P9. However, more technical respondents had a somewhat different view of the artifacts informing the problem domain. Examples being P3 stating that "*precise definitions of object relations are always a benefit*", and P6 expressing that the different artifacts are "*relevant for technical design*". There were however also respondents very familiar with the artifacts that still perceived them as not useful in upfront analysis and design. P12 is saying about the class diagram that it is "*only useful in very concrete situations*", and P13 stating that a class diagram "*should not and cannot be done before Sprint 1*".

Although respondents perceive the analysis and design artifacts as being less useful before the first Sprint, there is a tendency for this to be because of respondents not being familiar with them. However, there is a slight indication that more technical profiles perceive the analysis and design artifacts as useful.

Application Domain

Analysis and design artifacts informing the application domain are among the highest rated, and respondents agree that these artifacts are useful before the first Sprint. Respondents are further familiar with these artifacts. Particularly work modeling is perceived as being very useful by respondents; giving this analysis and design artifact an average rating of 4.31 out of 5. P2 saying that it is "*essential for developing a good solution*", and P10 stating that "*here you have the whole process described and what data to store*".

One analysis and design artifact is, however, dividing respondents. Personas are perceived as being very useful by some, and not useful at all by others. This can also be seen in Table 4, where personas have an average rating just under the median value. P2 is expressing that personas are "*essential for understanding who is going to use the solution*". Along the same lines, P5 says that "*personas are interesting as they give insights into who our users of the system are*". P1, however, says that "*personas in their theoretical form are completely useless. No one wants to read about a fictitious user*", and P4 stating that "*personas are okay, but one should be careful not to create archetypes where there is a risk of designing for a very small group*".

In summary, respondents are in general agreement that the analysis and design artifacts informing the application domain are useful before the first Sprint. One artifact diving respondents are personas, which is also apparent in the average perceived usefulness—placing personas just under the median value.

System Architecture

Respondents are somewhat divided on whether the artifact informing system architecture is useful before the first Sprint or not. This can also be seen by the average rating of the artifact, giving component architecture a rating of 2.85 out of 5 in total. Further, respondents are not so familiar with this artifact.

The open-ended question provides some more insights. P5, despite being only a little familiar with this analysis and design artifact, say that "*I've done technical architecture and it is a good idea before Sprints*". Other respondents are however of another opinion. P2 is stating that it is "*way to technical and solution specific before the Scrum-team starts*". Finally, some respondents rate the artifact very low but indicate that it is an analysis and design artifact that could prove useful under certain circumstances. P12 expresses that "*it is a fine activity - primarily for the technical profiles. But I don't know many teams designing so specific up front*".

Regarding system architecture, respondents are divided regarding the perceived usefulness. However, the open-ended responses indicate that some respondents perceive this analysis and design artifact as useful before the first Sprint—and others under certain circumstances. An interesting observation is that most respondents are not very familiar with the artifact specifying the component architecture.

Component Design

Nearly all respondents agree that prototypes are very useful before the first Sprint rating it 4.46 out of 5 on average. Likewise, all respondents are very familiar with this analysis and design artifact. They do not, however, perceive the model component as being useful before the first Sprint; rating it 2.46 out of 5.

P1 says that prototypes are "*the best design description there is*", P4 saying that "*prototyping is my favorite deliverable to* *customers and colleagues*", and P6 expressing that "*it provides a good overview and insights*". However, few respondents were more critical of using prototypes. P5 is saying that "*[...] they should be done as part of the development Sprints. A prototype should be done when there is doubt about the design or when you want to test a design*".

Regarding the model component, respondents agree that it is not very useful before the first Sprint. One exception being P3 stating that it is "*detailed and manageable*". Others point to it being an artifact belonging in later Sprints, or as documentation. An example being P13 who says that the model component can be used for "*documentation of limited areas of existing applications where it is not obvious from the code*".

Summarizing the analysis and design artifacts informing the component design, respondents all perceive prototypes as being very useful before the first Sprint. On the other hand, they perceive the model component as being not very useful.

ELEMENT 2: FINDINGS

Key results from the diary reflecting the practical relevance of selected artifacts covering categories of analysis and design are presented in the following. We used a qualitative, inductive analysis of the diary to find the total number of times each artifact was used during development, but also the reported benefits and drawbacks to using the artifacts. This was done in an iterative manner using NVivo.

During coding of the diary, we observed common themes related to the use of the selected upfront analysis and design artifacts; one of them being *tacit use*. Several times during the exploratory study, we have reflected upon decisions founded on the tacit knowledge of the bioanalysts at the hospital. This knowledge was generated as a result of developing the artifacts presented in this paper—thus tacitly relying on the artifacts without explicitly having them at hand. Therefore, we claim that the artifacts had practical relevance for developing the application despite not being consulted in a direct manner. The direct and tacit use of the different artifacts during each Sprint can be seen in table 5 on the following page.

High-level Description

Artifacts informing the high-level description of the intended system was used very sparsely; the system definition being used only once during development, and the affinity diagram not being used at all.

This does not, however, testify to the artifacts being irrelevant for the actual development of the system. For instance, the system definition provides a description of the intended system, which developers and relevant stakeholders can understand and appreciate. And so the system definition acts as guidelines for communication between developers and the Product Owner—even if the system definition is not consulted explicitly. In this case, the system definition provided the foundation for the prioritizing of User Stories on the Product Backlog. Before the first Sprint, two stakeholders from the Hospital acting as Product Owners were introduced to the general system definition of the intended system; and from there they prioritized the User Stories. As noted in the diary, after the introduction to the intended system "*we talked about which part of the [intended] functionality would be the best to start with*". Without having defined the intended use and functionality of the system, prioritizing the required functionality would, in this case, be very difficult.

Regarding the affinity diagram, it is hard to evaluate the practical relevance. The artifact was not directly or tacitly used as a foundation for making decisions during development—so in this sense, one could reach the conclusion that the artifact had no relevance for developing the system. One important observation, however, is that the exercise of developing an affinity diagram has the purpose of organizing and structuring data. And so informs other categories of analysis and design. From this point of view, the affinity diagram is very relevant to the development of other artifacts; however, it is not directly evident or referenced in the diary.

Problem Domain

Artifacts informing the problem domain was used on several occasions during development; both directly and tacitly. The class diagram was used 4 times, the event table was used once, and the statechart diagram was used 3 times in total.

During the first week of development, the artifacts informing the problem domain was used to verify the correctness of relationships between objects in the application. More specifically, after implementing the functionality of saving objects in the application to the database, the artifacts were used as a reference for how the relationships were in practice. Thus providing guidance and confirmation to the correctness of the proposed solution. After considering different ways of implementing the intended solution, the artifacts helped choose the one perceived as being most correct. As written in the diary, we observe that "*having modeled the problem domain satisfactorily, it is obvious that the requisition is the most important object*". The class diagram and the statechart diagram was used again in Sprint 3. Here the artifacts were again used as a confirmation of the correctness of the intended implementation of functionality. On one occasion, we contemplate different ways of modeling how a department at the hospital is associated with a specific Team of bioanalysts, and the class diagram and statechart diagram provides information resulting in the choosing of one solution over the other. Before deciding on the best possible solution, we note on the basis of the class diagram and statechart diagram, that "*it is in other words a necessity to model the Team in the system as a given department is always associated with one—and only one—Team*".

Summarizing, the artifacts informing the problem domain was used on multiple occasions during development. And further, the use of the artifacts was of practical relevance when doubting whether one solution would be preferable to the other.

Application Domain

Two of the artifacts informing the application domain was frequently used during development. Scenarios describing the intended system was used 5 times in total, and work modeling was used 10 times. However, personas were not used—either directly or tacitly—once.

		S1		S2		S3		S4		
		D	T	D	T	D	T	D	T	Total
High-level Description	System definition	-	1	-	-	-	-	-	-	1
	Affinity diagram	-	-	-	-	-	-	-	-	0
Problem Domain	Class diagram	1	-	-	-	1	2	-	-	4
	Event table	1	-	-	-	-	-	-	-	1
	Statechart diagram	1	-	-	-	-	2	-	-	3
Application Domain	Personas	-	-	-	-	-	-	-	-	0
	Scenarios	1	1	1	-	2	-	-	-	5
	Work modelling	1	4	-	-	-	5	-	-	10
System Architecture	Component architecture	-	-	-	2	-	-	-	-	2
Component Design	Prototypes	2	-	-	-	2	-	-	-	4
	Model component	6[1]	1	-	-	1[2]	-	1[3]	-	9

Table 5. Number of times each artifact was used directly (D) and tacitly (T) during development Sprints.

Scenarios were mainly used as input to the User Stories describing requirements. And so, scenarios were typically used directly in relation to the Sprint Planning. For example, we have noted that "*[scenarios] were developed to illustrate how the intended system would help the bioanalysts—and so they very much mimic User Stories*", and further that "*they were very useful when I was to validate User Stories*". Work modeling was the artifact used the most during development, however, an interesting observation is that it was used almost exclusively in a tacit way. The development of work models during upfront analysis and design made explicit different required usages of the intended system. One example being the need for scanning both the barcodes on the patient's bracelet and on the different test tubes. This needs to be done in specific order, and without needing to consult documentation or users, the development of work models tacitly provided the necessary information. As noted in the diary, we comment that "*I've implemented functionality that simulates scanning the patients and the test tubes' bar-codes. It is very simple, but provide the opportunity of [...] simulating the blood sampling*". Another example of the tacit use of work models is different changes made to the user interface based on tacit knowledge about the bioanalysts and their needs. Commenting on why the user interface was simplified during implementation, it is noted in the diary that "*[It] is obvious that the bioanalysts first orientate themselves towards the location of the patient, then confirms their name and social security number*", which is resulting in "*[...] hiding less important information [in the user interface]*". Personas were not used during development, and a suspected reason for this is that there was high-level of user commitment and interaction during development. As such, there was no need for referencing the typical user. Further, since the application was built as a form of custom product targeting a specific user group with the same required functionality, the need for abstracting information about potential users seems less important or useful.

In summary, the artifacts informing the application domain was frequently used during the development process—except personas that were not used once. Scenarios provided guidance and confirmation when developing User Stories, and work modeling was frequently used in a tacit way when developing required functionality or changing the user interface.

System Architecture

The artifact informing system architecture was used 2 times during development, and it was in relation to the implementation of functionality for communicating with a web server.

One of the requirements for the intended application was removing the need for physical requisitions—and instead store them digitally. Therefore, during upfront analysis and design, the component architecture artifact was developed to reflect how the application would communicate with a central server system and database. Following this thought, the application was built around the same principle. We have not noted any direct use of this artifact—however, the very idea of creating a client-server pattern with the possibility of having multiple devices communicating with the same central server system simultaneously is a direct consequence of this Design decision. And so, when developing this functionality, it is viewed as a tacit use of the artifact. For example, when it is noted that "*This request contains updated information about the requisition, and the web server is responsible for parsing and storing the data*", it is a direct consequence of the artifact developed during upfront analysis and design. Had the component architecture been developed differently—e.g. storing data locally on the device—the functionality would, in turn, have been implemented differently. It follows, then, that the artifact had a big influence in developing the system even though it was only used tacitly; and only used twice.

Component Design

Artifacts informing the component design of the application was frequently used during development; with prototypes being used 4 times, and the model component being used a total number of 9 times.

The prototypes developed during upfront analysis and design was used during development mainly in relation to the actual implementation of the user interface. In this regard, the prototypes were directly translated into the application. An example

[1]First revision of model component

[2]Second revision of model component

[3]Third revision of model component

being during Sprint 3, when implementing functionality for showing the different departments in the hospital. Here it is noted that "*[the] prototypes once again formed the basis for the user interface*". It is further commented that the "*prototypes have been a good starting point for developing the user interface, since they have been developed with a certain attention to detail that have been easy to copy in the actual application*", and that the prototypes "*have saved me from coming up with my own design—and evaluate what data to represent in the user interface*". The model component was used throughout the entire development process and had different practical uses. In the first Sprint, the model component was used most frequently, and here it served the purpose of validating the implementation of functionality and objects. We note that "*is has been particularly useful in relation to modeling the different objects in the application*" and that the model component was used directly when "*[implementing] the relations between Patient, Department, and Bed*[4]". However, one recurring theme is that the model component was revised multiple times during development—to reflect the current state of the application. In this sense, the model component served both as guidelines for further development of the application, but also as documentation. In the first Sprint, the author notes that the initial model component developed during upfront analysis and design was insufficient, and so chose to revise the model component as a way of abstracting information and managing increasing complexity. Further, when implementing a new object during Sprint 3, a revised model component was the deciding factor for how the different relations between objects were implemented. As we have noted, "*since the application is becoming somewhat complex [...] the only logical solution for me, was modeling the application by revising the existing model component*". This also helped in developing the database design even though no artifacts were developed to support this activity. It is commented that "*I haven't had material available that models the database—but in practice, I've been able to use the model component to such a degree, that it haven't been necessary*". And further, that the revision of the model component resulted in an application architecture that "*is forthcoming to the required functionality—even as the application develops and becomes more complex*".

Summarizing the artifacts informing component design, both prototypes, and the model component was frequently used during development. The paper prototypes were translated to the application and guided the development of the user interface. The model component served as both documentation, and confirmation of decisions regarding implementation.

DISCUSSION

This section discusses and compares the findings of both perceived and practical relevance of the selected artifacts covering possible categories of upfront analysis and design.

Differences in perceived and practical relevance

The survey investigating the perceived relevance of selected artifacts indicate that respondents do not value artifacts modeling the problem domain, or the model component that serves as implementation details of the problem domain model. However, these artifacts—and particularly the model component—was some of the artifacts used the most during the exploratory study.

The respondents who perceive the artifacts as being not useful before the first Sprint, do so for mainly two different reasons. The first being that they are unfamiliar with the artifacts, and the second being that they find it too specific too early in the development process. The latter also reflecting how modeling maybe is considered part of a plan-driven approach and BDUF. Yet, the practical use of the artifacts tell a different story. In that regard, we benefited from the artifacts—and frequently note how the structuring and abstracting of information helped built a better application, and helped manage increasing complexity as the application developed. An important observation is, however, that the model component—which was used most frequently—was not considered final or unchangeable during development. On the opposite, the model component evolved along with the application, always reflecting the current state of the application. This way the model component not only served as reference when implementing new functionality, and the correctness thereof, but also as documentation of previous design decisions. Further, had multiple developers been working on the same application, an iterative approach to modeling the application could prove valuable in communicating Design decisions and potential pitfalls of the current implementation.

Another more discrete difference lays in the perceived and practical relevance of the system architecture. Respondents were divided on the perceived usefulness—some finding it relevant for the technical profiles, and others stating that they find it too specific before the first Sprint. However, the use of the system architecture proved to have a strong influence on the system design. The component architecture modeled the application in such a way, that data would be stored and managed on a central server; allowing the application to communicate via a client-server pattern. This provided the foundation for the entire application, and all functionality was developed with this in mind. Had the component architecture been developed differently, or had such considerations not been addressed before development, the development of the application would have proceeded differently. However, it is also worth noting that the architectural considerations could have been documented in other ways than by modeling the component architecture. For instance, considerations regarding architectural patterns could have been expressed in natural language, or as part of the vision for the intended system; as some respondents also indicate when they express the need for specifications in the system definition.

Similarities in perceived and practical relevance

The survey respondents perceived work modeling and prototypes as being the most useful artifacts, and consequently these artifacts were also among the most frequently used during the exploratory study. Further, the system definition specifying the intended solution have both perceived and practical relevance.

The respondents ascribe value to work modeling, generally indicating that it provides a detailed process view, and helps

[4]Objects in application

visualize how the intended solution should be developed. The same theme can be found in the use of work modeling during development of the application, where we attribute many Design and implementation decisions to the tacit use of work modeling specifying the bioanalysts work practice. Had this artifact not been developed, much of the functionality in the application would likely have been implemented differently.

The prototypes were described by some respondents as the best design description possible, and the practical use of the prototypes confirms this view. During development, the application's user interface was based on the prototypes—and they were used whenever new elements were added to the user interface. Thus being highly relevant for the intended solution, and influencing the development of the application.

Lastly, the vision of the intended system expressed as a prosaic system definition was perceived as being useful by respondents, and the practical use of the system definition helped stakeholders understand and prioritize requirements on the Sprint Backlog. Although the system definition was only directly used once during development, the practical relevance of the system definition proved to be significant. It shaped the application, and was a medium for communication between the developer and the Product Owners.

Importance of tacit use

An interesting observation following the analysis of the development diary, is the importance of tacit use of analysis and design artifacts.

If we were to consider practical relevance of the selected artifacts based solely on their direct use, the overall impression of relevance of artifacts presented in this paper would be very different. We have observed how the development of artifacts resulted in tacit knowledge about e.g. work practice or the problem domain, and that this influenced decisions during development. Comparing the direct and tacit use of artifacts, the tacit use of selected artifacts account for 46% of the total number of times the artifacts were used during development—a testament to the importance of the knowledge generated as a result of developing the artifacts.

We are, however, aware that other circumstances likely would produce different results with regards to the importance of tacit use of artifacts. An example being a larger development team, where all developers would not have the opportunity of directly interacting with users. In this case, one would assume that the distribution of direct and tacit use would be very much different—since most developers would not have been part of developing upfront analysis and design artifacts. Thus, artifacts would most likely be used in a direct way most of the time. However, we claim that the tacit use of artifacts observed in this paper still attributes to the practical relevance of developing analysis and design artifacts in Scrum.

Context dependence

We cannot overlook context when developing software. And in this case, the specific case could have influenced both the perceived and practical relevance of selected artifacts.

The respondents were introduced to the case in the survey, and it is possible that the case description have influenced their responses. According to [25], a development situation is characterized by varying degrees of complexity and uncertainty; each warranting different countermeasures. A high degree of complexity requires abstraction and decomposition, while uncertainty calls for an experimental approach. Thus, if respondents perceived the case as being characterized mainly by one or the other, their responses could reflect this perception. The same can be said regarding practical relevance, where another case characterized by other degrees of complexity and uncertainty could yield different findings.

Proposed artifacts

Practitioners were given the opportunity to recommend further analysis and design artifacts not part of this empirical study. We provided the opportunity to bring further perspectives on Design Upfront when developing with Scrum; however, only a few respondents gave their recommendations. In this section, we will recap on the relevant suggestions from practitioners.

Two suggestions were repeated in the responses. Both P1 and P13 recommend the use of sequence diagrams to model important or complex communication between components in the intended system. This is interesting, because even though respondents refer from artifacts informing implementation details (such as the model component)—this suggestion points to further value in specifying system components.

The other suggestion repeated in the responses concerned modeling of the usage of the system. P1 and P6 both indicate that an artifact such as storyboards helps in visualizing the intended system—and how users interact and benefit from the system. Storyboards would in this study inform the application domain, and we find that it could be a possible replacement for artifacts such as personas and scenarios; seeing as storyboards would provide information about both users and intended use of the system.

Practical implications

Following findings from the empirical study, and a discussion thereof, we will in closing highlight some practical implications.

Choose artifacts systematically

Analysis and design artifacts in Scrum should be developed with a clear purpose in mind; especially since artifacts, as we have shown, address specific aspects of analysis and design. Practitioners should therefore make clear what the purpose of the given artifacts are, and choose sensibly.

Value simplicity of artifacts

Analysis and design artifacts in Scrum should be developed with the value of simplicity in mind. As we have demonstrated in our empirical study, there is practical relevance in using artifacts in an iterative manner—and accepting that artifacts may well develop along with the IT system.

Address possible aspects of analysis and design

Practitioners should acknowledge the importance of both system-focused and user-focused design. Our empirical study show that artifacts covering possible aspects of analysis and design to a certain degree have both perceived and practical relevance. Therefore, artifacts should address different, possible aspects of analysis and design.

CONCLUSION

This paper has presented an empirical study investigating the perceived and practical relevance of analysis and design artifacts when developing with Scrum. The empirical study consisted of two different elements. The first element being an empirical survey of agile practitioners' perceived relevance of selected analysis and design artifacts. The second being an exploratory study investigating the practical relevance of the same analysis and design artifacts.

Comparing the findings of the two elements in our study, we find that there are both differences and similarities in the perceived and practical relevance of selected upfront analysis and design artifacts. Artifacts modeling the system was perceived to be less relevant in the empirical survey, but highly relevant during development. On the opposite, prototypes, work modeling, and the system definition proved to have both perceived and practical relevance. Lastly, we identify two different usages of artifacts during development: Direct and tacit use. Several times during development, decisions were made based on tacit knowledge generated as a result of developing upfront analysis and design artifacts.

In closing, we highlight practical implications based on our empirical study. Here we propose to choose artifacts systematically, value simplicity of artifacts, and address possible aspects of analysis and design.

Limitations

We acknowledge the limitations of our work, and especially the nature of case studies. The perceived and practical relevance of selected analysis and design artifacts could prove to be different in other contexts. Further, we recognize that the survey presented in this paper have a rather limited number of respondents. We have attempted to negate this by targeting a very specific population, and by providing open-ended questions to provide more insights into the responses.

Future work

Additional studies should be performed to investigate the relevance of other analysis and design artifacts in ASD. And further, empirical studies should be performed to investigate the use of artifacts in larger, agile teams.

Lastly, empirical studies investigating the relevance of artifacts in different development situations could provide valuable insights to practitioners. Such empirical studies could provide recommendations as to which artifacts have practical relevance dependent on whether a development situation is characterized mainly by a high degree of complexity or uncertainty.

ACKNOWLEDGMENTS

We would like to thank all participants from Clinical Biochemistry at Aalborg University Hospital for being very forthcoming, and for their contribution to this empirical study. Further, we would like to thank Robin Damsgaard Larsen and Mikkel Mørch Henriksen for their help in conducting part of the literature survey, and in developing the selected analysis and design artifacts.

REFERENCES

1. Scott Ambler. 2002. *Agile modeling: effective practices for extreme programming and the unified process*. John Wiley & Sons.

2. Alvaro Aranda Muñoz, Karin Nilsson Helander, Thijmen de Gooijer, and Maria Ralph. 2016. Integrating Scrum and UCD: Insights from Two Case Studies. In *Integrating User-Centred Design in Agile Development*, Gilbert Cockton, Marta Lárusdóttir, Peggy Gregory, and Åsa Cajander (Eds.). Springer International Publishing, Cham, 97–115. http://dx.doi.org/10.1007/978-3-319-32165-3_4

3. Giuliano Armano and Michele Marchesi. 2000. A Rapid Development Process with UML. *SIGAPP Appl. Comput. Rev.* 8, 1 (Sept. 2000), 4–11. DOI: http://dx.doi.org/10.1145/361651.361653

4. Muhammad Ali Babar. 2009. An exploratory study of architectural practices and challenges in using agile software development approaches. In *2009 Joint Working IEEE/IFIP Conference on Software Architecture European Conference on Software Architecture* (2009-09). 81–90. DOI: http://dx.doi.org/10.1109/WICSA.2009.5290794

5. Kent Beck. 2000. *Extreme programming explained: embrace change*. Addison-Wesley Professional.

6. Hugh Beyer. 2010. User-Centered Agile Methods. *Synthesis Lectures on Human-Centered Informatics* 3, 1 (2010), 1–71. DOI: http://dx.doi.org/10.2200/S00286ED1V01Y201002HCI010

7. Barry Boehm. 2002. Get ready for agile methods, with care. *Computer* 35, 1 (2002), 64–69. DOI: http://dx.doi.org/10.1109/2.976920

8. Silvia Bordin and Antonella De Angeli. 2016. Focal Points for a More User-Centred Agile Development. In *Agile Processes, in Software Engineering, and Extreme Programming*, Helen Sharp and Tracy Hall (Eds.). Number 251 in Lecture Notes in Business Information Processing. Springer International Publishing.

9. 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 (2015), 163–181. DOI: http://dx.doi.org/10.1016/j.infsof.2015.01.004

10. Michael Budwig, Soojin Jeong, and Kuldeep Kelkar. 2009. When User Experience Met Agile: A Case Study. In *CHI '09 Extended Abstracts on Human Factors in Computing Systems (CHI EA '09)*. ACM, New York, NY, USA. DOI: http://dx.doi.org/10.1145/1520340.1520434

11. Stephanie Chamberlain, Helen Sharp, and Neil Maiden. 2006. Towards a Framework for Integrating Agile

Development and User-Centred Design. In *Extreme Programming and Agile Processes in Software Engineering*, Pekka Abrahamsson, Michele Marchesi, and Giancarlo Succi (Eds.). Number 4044 in Lecture Notes in Computer Science. Springer Berlin Heidelberg, 143–153.

12. Tore Dybå and Torgeir Dingsøyr. 2008. Empirical studies of agile software development: A systematic review. *Information and Software Technology* 50, 9 (2008), 833–859.

13. Davide Falessi, Giovanni Cantone, Salvatore Alessandro Sarcia, Giuseppe Calavaro, Paolo Subiaco, and Cristiana D'Amore. 2010. Peaceful coexistence: Agile developer perspectives on software architecture. *IEEE software* 27, 2 (2010).

14. Martin Fowler. 2001. Is design dead? *SOFTWARE DEVELOPMENT-SAN FRANCISCO* 9, 4 (2001), 42–47.

15. Martin Fowler and Jim Highsmith. 2001. The agile manifesto. *Software Development* 9, 8 (2001), 28–35.

16. Jeff Gothelf and Josh Seiden. 2013. *Lean UX: Applying lean principles to improve user experience*. "O'Reilly Media, Inc.".

17. Jan Gulliksen, Inger Boivie, Jenny Persson, Anders Hektor, and Lena Herulf. 2004. Making a difference: a survey of the usability profession in Sweden. In *Proceedings of the third Nordic conference on Human-computer interaction*. ACM, 207–215.

18. Zahid Hussain, Wolfgang Slany, and Andreas Holzinger. 2009. Current state of agile user-centered design: A survey. In *Symposium of the Austrian HCI and Usability Engineering Group*. Springer, 416–427.

19. Leif Obel Jepsen, Lars Mathiassen, and Peter Axel Nielsen. 1989. Back to thinking mode: diaries for the management of information systems development projects. *Behaviour & Information Technology* 8, 3 (1989), 207–217. DOI: http://dx.doi.org/10.1080/01449298908914552

20. Yong Gu Ji and Myung Hwan Yun. 2006. Enhancing the minority discipline in the IT industry: A survey of usability and User-Centered design practice. *International Journal of Human-Computer Interaction* 20, 2 (2006), 117–134.

21. Yuan Jia, Marta Kristin Larusdottir, and Åsa Cajander. 2012. The Usage of Usability Techniques in Scrum Projects. In *Human-Centered Software Engineering*, Marco Winckler, Peter Forbrig, and Regina Bernhaupt (Eds.). Number 7623 in Lecture Notes in Computer Science. Springer Berlin Heidelberg, 331–341.

22. Kati Kuusinen. 2014. Beyond the "One Sprint Ahead" Approach: Organizing User Experience Work in Agile Software Development. In *Workshop at NordiCHI*. https://ucdandagile.wordpress.com/papers/

23. Marta Larusdottir, Jan Gulliksen, and Åsa Cajander. 2016. A license to kill – Improving UCSD in Agile development. *Journal of Systems and Software* (2016). DOI:http://dx.doi.org/10.1016/j.jss.2016.01.024

24. Lars Mathiassen, Andreas Munk-Madsen, Peter Axel Nielsen, and Jan Stage. 2000. *Object-oriented analysis & design*. Citeseer.

25. Lars Mathiassen and Jan Stage. 1992. The Principle of Limited Reduction. *Information Technology and People* 6 (1992), 171–185.

26. Maryam Najafi and Len Toyoshiba. 2008. Two Case Studies of User Experience Design and Agile Development. In *Agile 2008 Conference*. 531–536.

27. Christian R. Prause and Zoya Durdik. 2012. Architectural design and documentation: Waste in agile development?. In *2012 International Conference on Software and System Process (ICSSP)*. 130–134. DOI: http://dx.doi.org/10.1109/ICSSP.2012.6225956

28. Steven Rakitin. 2001. Manifesto elicits cynicism. *IEEE Computer* 34, 12 (2001), 4.

29. Outi Salo and Pekka Abrahamsson. 2008. Agile methods in European embedded software development organisations: a survey on the actual use and usefulness of Extreme Programming and Scrum. *IET software* 2, 1 (2008), 58–64.

30. Thiago Silva da Silva, Angela Martin, Frank Maurer, and Milene Silveira. 2011. User-Centered Design and Agile Methods: A Systematic Review. In *Agile Conference (AGILE), 2011* (2011-08). 77–86. DOI: http://dx.doi.org/10.1109/AGILE.2011.24

31. Mona Singh. 2008. U-SCRUM: An Agile Methodology for Promoting Usability. In *Agile 2008 Conference*. 555–560. DOI:http://dx.doi.org/10.1109/Agile.2008.33

32. Desirée Sy. 2007. Adapting Usability Investigations for Agile User-centered Design. *J. Usability Studies* 2, 3 (2007), 112–132.

33. Jim Ungar and Jeff White. 2008. Agile user centered design: enter the design studio-a case study. In *CHI'08 Extended Abstracts on Human Factors in Computing Systems*. ACM, 2167–2178.

34. Karel Vredenburg, Ji-Ye Mao, Paul W Smith, and Tom Carey. 2002. A survey of user-centered design practice. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 471–478.

35. Heather Williams and Andrew Ferguson. 2007. The UCD Perspective: Before and After Agile. In *Agile Conference (AGILE), 2007*. 285–290. DOI: http://dx.doi.org/10.1109/AGILE.2007.61

36. Alan Wills. 2001. Catalytic Modeling: UML meets XP.. In *pUML*. 288–306.

37. Chen Yang, Peng Liang, and Paris Avgeriou. 2016. A systematic mapping study on the combination of software architecture and agile development. *Journal of Systems and Software* 111 (2016), 157–184. DOI: http://dx.doi.org/10.1016/j.jss.2015.09.028

38. Tina Øvad and Lars Bo Larsen. 2015. The Prevalence of UX Design in Agile Development Processes in Industry. In *Agile Conference (AGILE), 2015*. IEEE, 40–49. DOI: http://dx.doi.org/10.1109/Agile.2015.13

An Exploratory Study of Direct User Involvement and Design Decisions in Scrum Sprints

Nicklas Holm Joergensen
Aalborg University
Department of Computer Science
Aalborg, Denmark
nhjo10@student.aau.dk

ABSTRACT

Scrum is one of the most widely used agile methods in industry and promises a framework for developing and sustaining complex software. While Scrum focus on delivering working software quickly, it does not focus on delivering usable software. Thus, practitioners try to find relevant ways of achieving a high degree of user involvement in Scrum with the long-term goal of arriving at effective strategies for relevant user involvement in Sprints. This paper presents an exploratory investigating what direct user involvement is relevant for in relation to Design decisions and Design process—and to what extent direct user involvement the influence development process' contingencies. We find that direct user involvement is relevant for defining the satisficing behavior of the developer; adjusting aspiration levels; and in transferring knowledge from user to developer. Further, we find that direct user involvement was relevant for validating and generating requirements during the Design process. Lastly, we find that direct user involvement influenced both uncertainty and complexity as contingencies during development.

Author Keywords

User-Centered Design; Scrum; agile development; direct user involvement

INTRODUCTION

Agile software development (ASD) encapsulates different methods all characterized by being light-weight and flexible [6], with Scrum being one of the most widely used in industry [12, 25]. Scrum promises a framework for developing and sustaining complex software that enables developers to further employ various processes and techniques [19].

While Scrum is a framework that focuses on delivering working software quickly, it does not focus on explicitly delivering *usable* software—neglecting the difference between the customer and the user [1]. As IT systems become more and more complex, developing user-friendly systems can help in achieving economic success in a competitive market [15] and result in increased customer satisfaction [8]. Thus leading practitioners to experiment with effective and relevant ways of achieving a high degree of user involvement in Scrum to develop better software.

One strategy for establishing user focus in Scrum is to organize development in two different *tracks*, where one track is concerned with technical implementation and development, and the other is concerned with user involvement; the so-called 'One Sprint Ahead' [22]. However, this is not without challenges and drawbacks. The separation of design and implementation increase costs of iterating over findings from user testing, and it prolongs the duration of Sprints which in turn makes it more cumbersome to react to changes during development [10].

A potential solution to the aforementioned challenges is to promote direct user involvement in Scrum Sprints. We are, however, still faced with challenges related to the evaluation of relevant user involvement activities; and deciding on what to focus on when interacting with users of the intended system. In some cases, it may be preferable to evaluate the user interface, while in other cases, it may be more beneficial to inquire about the daily work practice. Empirical studies suggest that some problems related to user involvement in Scrum stem from bypassing user-focused activities because of limited time and resources [11]. Hence, a possibly more beneficial strategy is to involve users only when it is deemed relevant, and only in a relevant manner.

A challenge and long-term goal for practitioners is therefore to arrive at effective strategies for deciding when and how to involve users in Sprints. As suggested by [3], empirical insights could provide practical advice on context-specific user involvement strategies. However, there is lacking empirical studies investigating the extent and outcome of direct user involvement in agile methods [3]. To contribute to this research gap, we find that we must first investigate and understand the influence of direct user involvement in Sprints on developers' Design decisions—and what this direct user involvement is relevant for during development. Further, we must investigate how direct user involvement influence possible contingencies in the development process to eventually arrive at recommendations for relevant user involvement in Sprints.

This paper presents an exploratory study investigating what direct user involvement is relevant for in relation to Design decisions and Design process—and to what extent direct user involvement influence the development process' contingencies. While developing an Android application for supporting the daily work of bioanalysts of the local university hospital, we kept a development diary documenting the development process, Design decisions, and how direct user involvement in Scrum Sprints influenced the development of the application.

The rest of this paper is organized as follows: The next section provides an overview of the theoretical foundation for this study and related work. Then we introduce the research method used in this exploratory study. The following section then presents findings from the study. Finally, we provide a discussion and conclusion.

RELATED WORK

In the following, we will first present our theoretical understanding of human problem solving and decision making. We will then present an overview of related work on developers' Design decisions and Design process.

Bounded rationality

Traditional theories of human problem solving and decision making has been concerned with global rationality and the idea of *the economic man*—however, Simon [20] opposes this ideal. This rational actor was assumed to have a vast knowledge of his environment, and being capable of calculating the optimal course of action among many different possibilities. Simon objects to this saying that the actor is constrained both temporally and cognitively. Instead, Simon proposes the theory of bounded rationality and states that:

> [T]he task is to replace the global rationality of economic man with a kind of rational behavior that is compatible with the access to information and the computational capacities that are actually possessed by organisms, including man, in the kinds of environment in which such organisms exist [20].

In doing so, Simon argues that actors are constrained by psychological limits which result in the actor simplifying information; and therefore relying on approximations. We are nowhere capable of performing the computations needed for choosing among a vast variety of actions and outcomes as suggested by the economic ideal. Thus, the problem-solving activity and the decision making process is about the actor approximating a solution that is *good enough*.

This is not to suggest that the actor is irrational in his behavior. It is to suggest that the actor while attempting to behave rationally, is still constrained by his environment and his cognitive capabilities. Simon introduces the concept of *satisficing* which encapsulates this behavior. When faced with several courses of action, the actor does not try to calculate the outcome of every possible course of action in search of the optimal decision. Instead, the actor chooses the first course of action meeting his aspiration levels [21].

The other possible approach that an actor can choose, when facing complexity and uncertainty in his problem solving or decision making, is optimization. However, this approach is characterized by being an approximation of optimization [21]. Thus, the actor tries to simplify the world until he reaches a situation that is manageable—and can choose the optimal decision based on this radically simplified real-world situation. There is a fine line between satisficing and approximate optimization; however, Simon contends that the practical implications of choosing one approach over the other can be very significant [21].

In one situation, the actor may start out with an approximate optimization of the problem-solving space—working with highly simplified models. When a plan has been schematized based on this simplified space, the actor may then introduce details to guide his satisficing design strategy. In another, the actor may employ an overall satisficing design strategy; filling in the details utilizing an approximate optimization approach [21].

User involvement and Design decisions

There are to the best of our knowledge no empirical studies investigating direct user involvement and what it is relevant for in relation to developers' Design decisions and Design process. However, other empirical studies have investigated developers' decision making and reasoning in the Design and development process.

Kant [9] performed a study with computer science students, investigating their process of designing an algorithm for computational geometry. They find that the participants typically develop a rapid core idea which they then refine as they go along. A drawback to this study, however, is that the problem students solved were complex because of a requirement for cleverness—and not because of an abundance of information to structure and process. This meaning that there may be differences concerning more abstract software Design.

Zannier et al. [24] did a multi-case study of 25 software designers, asking about Design decisions they made. The study presents findings from interviews with the software designers, and provide some insights into developers' decision making. One is that software design often is related to the structuring of problems and that the structure of the problem influences the decision-making approach. Another that software designers often use satisficing behavior when evaluating a design. The study does not, however, offer any insights into the role of user involvement and software designers' decision making.

Lastly, Tang et al. [23] did an empirical study with both software professionals and students, investigating how much reasoning and exploration they performed before making architectural design decisions. It was conducted as a scenario-based questionnaire with 72 participants; however, only 11 of the participants filled out a questionnaire afterward about their reasoning process and providing further insights. The study concludes that a majority of both professionals and students exhibited satisficing behavior in their decision making and that this could impact the quality of Design decisions.

Summarizing, there are empirical studies investigating developers' decision making and their reasoning about Design decisions. Further, empirical studies point to developers exhibiting satisficing behavior when faced with Design decisions. There are, however, no empirical studies on direct user involvement and how this influences developers' decision making. If we are to reach a long-term goal of developing effective user involvement strategies, we first have to understand in what ways direct user involvement in Sprints are relevant for Design decisions and Design process.

METHOD

This paper aims at providing empirical evidence on what direct user involvement is relevant for in relation to Design decisions and Design process. We do this by conducting a qualitative case study of an exploratory nature. While developing an Android application for supporting the daily work of bioanalysts at the local university hospital, we kept a development diary documenting the development process and the influence of direct user involvement. In the following, we will first present the case forming the basis of this exploratory study and the participants. Then we present our data collection, and finally, we describe our process of data analysis.

Case

During this exploratory study, we have collaborated with Department of Clinical Biochemistry at Aalborg University Hospital. The department is responsible for taking and analyzing blood samples from patients admitted to the hospital, and further handling samples from local medical practitioners delivered to the hospital. Amounting to more than a thousand daily blood samples on average.

The staff at the department carry around multiple different devices to support their work practice; e.g. multiple telephones, a PDA scanner, and a pager (see Figure 1). This means carrying around several kilograms of digital equipment, even though none of the devices have capabilities not found in a regular smartphone. Further, the department experienced a lot of difficulties in their work practice that the devices did not support. One example is the difficulty in locating the other bioanalysts when offering assistance. Bioanalysts at the hospital are assigned different Teams before shifts, which dictates which departments to service at the hospital—and this means that not all Teams have the same number of blood samples to take each day. And so, typically, bioanalysts at the hospital offer to help each other in finishing bloodwork of patients if they are finished early with their immediate tasks. In this case, one would have to manually call co-workers not knowing where they are at the hospital, or if they are even in need of assistance with their immediate tasks; which was experienced as being very time-consuming and frustrating.

Therefore, we decided in collaboration with the staff at the department, to develop an Android prototype that would consolidate the different devices into one, single device. And even more so, develop a possible solution to the problem of locating and assisting co-workers. Thus making the daily work of the bioanalysts at Clinical Biochemistry easier and more convenient.

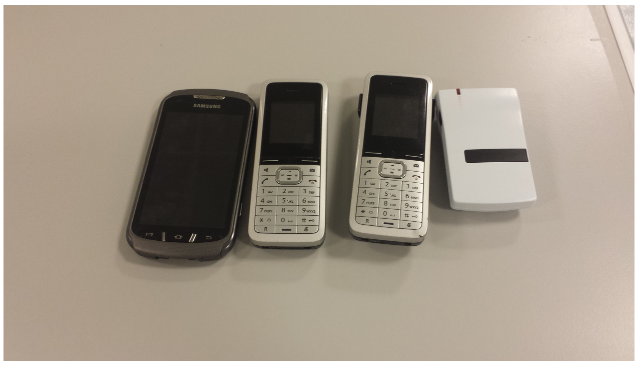
Figure 1. An example of devices typically carried by a single bioanalyst during a normal shift.

Participants

The study was conducted with six volunteering staff from Clinical Biochemistry at Aalborg University Hospital. The participants had different responsibilities and positions at the hospital and were chosen to represent different roles in the development process. A summary of participants and designated roles during development can be seen in Table 1.

The first two participants were chosen to represent the Product Owners, and thus being responsible for prioritizing Tasks on the Backlog. Product Owner 1 is working as a biochemist at the hospital, responsible for quality assurance and research; Product Owner 2 is working as a laboratory professional in close contact with the bioanalysts at the hospital.

Four more staff at the hospital further participated in the development process; we call them User 1-4. User 1 was chosen to be the one directly involved in Sprints, as he is working as a bioanalyst at the hospital, and is further responsible for educating new bioanalysts at the hospital. Thus, it was decided that this participant would be a very suited representative of the intended users; having an excellent understanding of not only work practice but also the challenges and requirements of the bioanalysts.

Lastly, User 2-4 volunteered to try the prototype immediately after the last Sprint. This was both to showcase and validate the final prototype. User 2-4 all work as bioanalysts at the hospital, and all have several years of experience as health professionals.

No.	Designation	Position
1	Product Owner 1	Biochemist and researcher
2	Product Owner 2	Laboratory professional
3	User 1	Bioanalyst and educator
4	User 2	Bioanalyst
5	User 3	Bioanalyst
6	User 4	Bioanalyst

Table 1. Overview of participants.

Data collection

The empirical data was collected via a diary kept during the development of the application spanning a period of three months in total. The use of diaries in other fields of research is very common [4]—and it has seen some use in the field of software development also [16].

There are several benefits to using diaries, and especially concerning an exploratory study. The primal benefits being the collection of very rich data [13], and that diaries tend to provide more reliable data in cases where estimates of frequency or temporal recollection are important [4]. However, there are also drawbacks to diaries; specifically the time-consuming nature of diaries resulting in "a process of attrition", and the failure to accurately recollect events if not recorded shortly after [4].

As countermeasures to the drawbacks of using diaries, we have followed the practical advice suggested in [7]. This resulting in defining a clear structure and purpose. Further, diary entries were written immediately after each day of development in an electronic format to strengthen the validity of recollection.

Data analysis

As with most qualitative studies, the starting point of analysis was coding and categorizing data. Coding involved multiple steps presented in the following. After initial coding, we derived themes based on reviewing the coded data presented in the findings section.

The first step in our data analysis was to thoroughly read through the data, familiarizing ourselves with sections and parts of the diary relevant to the research focus. After doing this, we performed what is referred to as *descriptive coding* [18]. The benefit to descriptive coding is that it creates an index of the data, making it easier to develop an overview of the content. These initial codes were developed in iterations, and finally reviewed and combined to create categories based on similar events and actions. Some were deduced directly from the research focus (e.g. *Design decisions*)—while others were established inductively through analysis (e.g. *Uncertainty*).

After developing the overall categories, we performed analytic rather than descriptive coding. This time focusing more on processes and properties related to the established categories than describing the events. After analytic coding of the data, codes were organized, reviewed, and combined to develop themes related to the categories developed from the descriptive coding of the data.

The diary was originally written in Danish, and coded using NVivo software. All example passages shown in this paper have been translated verbatim into English.

DIRECT USER INVOLVEMENT EPISODES

This section provides an overview of the different episodes of direct user involvement during Sprints to contextualize our findings. Table 2 on the following page shows a summary of all episodes.

Episode 1

The first episode took place right before the first Sprint and involved the two Product Owners from the hospital. The meeting had two purposes; (1) introducing and discussing the intended application and (2) prioritization of the developed User Stories on the Product Backlog. After discussing the perceived needs and requirements of bioanalysts at the hospital, the two Product Owners prioritized User Stories—specifying what would be the best functionality to develop and evaluate first.

Episode 2

The second episode took place at the hospital with User 1 immediately after the first Sprint, and the meeting took approximately one hour. Before the meeting, we had prepared questions regarding uncertainties discovered during the first weeks of development; pertaining mostly to work practice and the user interface. During this meeting, User 1 also tested the first functionality and provided inputs to both the user interface and lacking functionality. These inputs and perceived improvements were added to the Backlog to be implemented in Sprint 2.

Episode 3

The third episode took place at the hospital with User 1 following the second Sprint, and the meeting took approximately one hour. The meeting had two main purposes; (1) validating the proposed changes to the user interface implemented since the last evaluation, and (2) to clarify the work practice of dividing psychical blood sample requisitions between bioanalysts. During the meeting, we first performed an informal evaluation of the prototype with User 1—and afterward, the task of dividing physical requisitions was clarified via a conversation with User 1; documented by written notes of our understanding of the work practice.

Episode 4

The fourth episode concluded the fourth and final Sprint. It took place at the hospital with User 1 immediately after the fourth Sprint and took approximately one hour. The meeting was performed as an informal evaluation showcasing the newest functionality and recent changes to the user interface. The focus was mostly on functionality since this was aimed at directly supporting the work practice of the bioanalysts; and more specifically, solving the problem of how to effectively assists colleagues.

Episode 5

The fifth and final episode took place at the hospital with User 2-4 following the fourth Sprint. The purpose was to showcase the prototype and give the bioanalysts a sense of how a possible solution could look. Further, the bioanalysts were encouraged to test the prototype's functionality, and give their reflections on how well it would support their work practice. This feedback was audio recorded for further perspectives on the usefulness of the developed prototype.

FINDINGS

In the following, we will present the major themes related to the Design decisions, Design process, and development process contingencies derived from the coded data. First, we present findings on what direct user involvement in Sprints was relevant for in relation to Design decisions. We then describe how direct user involvement contributed to generation and validation of requirements as part of the iterative Design

No.	Participants	When	Purpose
1	Product Owners	Before Sprint 1	Introduction to the intended application, and prioritization of User Stories on the Product Backlog.
2	User 1	After Sprint 1	Showcase first iteration and clarify uncertainties generated in the first Sprint.
3	User 1	After Sprint 2	Informal evaluation of prototype and changes implemented since Episode 2. Clarify work practice.
4	User 1	After Sprint 4	Conclude development Sprints and informal evaluation of prototype.
5	User 2-4	After Sprint 4	Showcase prototype and functionality to further users at the hospital.

Table 2. Episodes of direct user involvement during Sprints

process. Finally, we present our findings on direct user involvement concerning complexity and uncertainty during the development of the application.

Design decisions

Developing and designing software involves a lot of problem-solving and decision-making, and the coded data provide some insights on how direct user involvement influenced Design decisions during development, and, consequently, in what ways this was relevant. In this section, we present findings related to satisficing behavior, aspiration levels, and knowledge transfer.

Defining satisficing behavior

The idea of satisficing among developers have been shown in different empirical studies, and we find that satisficing behavior exhibited from both developer and user played a significant role in development also. In the following passage, for example, it can be seen that when comparing different possible solutions for the relationship between tables in the database, one is chosen because of it being the solution seemingly meeting the needs and requirements of the developer.

> I [the developer] have chosen to keep this relation [between tables] since it still seems to be the best solution. I cannot, as of writing, see how else it would make sense to create the relation between a User[1] and a Requisition[2].

Few attempts had been made to structure the application differently leading up to the decision—however when discovering what was seemingly the right choice; the search strategy was terminated. Another example of satisficing behavior can be seen in the following passage. Here, a decision was made to implement a certain Object Relational Mapping library after comparing few solutions.

> After reading different comparisons [of libraries], documentation [...] I ended up with GreenDAO[3], as it seems to fulfill my needs—and has good backing.

In this case, also, the decision to choose one library over the other was made when finding the first solution best meeting needs and expectations; not by comparing and investigating every possible course of action. In the example passages above, the satisficing behavior of the developer was not specifically influenced by direct user involvement—but was more a result of the technical implications of developing the required functionality.

[1]Table representing object in application
[2]Table representing object in application
[3]Object Relational Mapping library for Android

However, direct user involvement did have an influence on the satisficing behavior of the developer; and especially with regards to the user interface. Take, for example, the following passage showing reflections about possible changes to the user interface after evaluating the first iteration with the User 1 during Episode 2.

> [...] much of the information on the physical requisition is required according to the hospitals' rules—however, in practice they are rarely used. The idea of hiding information as much as possible in the user interface, therefore, seems [to the user] like a really good solution.

In this case, considerations of a particular user interface design are derived from the direct interaction with User 1. Here, it is not necessarily the developer's satisficing behavior that guides the search of potential solutions; but rather the user's. An implication of this is also, that while the developer is working under certain cognitive constraints—parts of the decision making can be handed over to the user where relevant. Thus reducing the information to be processed and structured.

An interesting observation, however, is that not all Design decisions were made as a result of satisficing behavior from only the developer—or only the user. In some cases, the satisficing decision of the developer was fine-tuned to satisfy the user following evaluation. As an example, take the following extract from the diary commenting on the decision of adding labels to the user interface instead of plain text.

> On a physical requisition it is written in text what color test tube to do the test in [...], but we agreed that it is probably more intuitive, that this is specified with a colored label or colored icon.

Before evaluating with User 1 during Episode 2, the decision of specifying the type of test tube in written text—as it is done at the hospital at the time of writing—had been made; seeing this as the appropriate and meaningful solution. However, user involvement directly influenced the final Design decision, seeing as the proposed solution was satisficing for the developer; but not the user. In this sense, the developer's satisficing behavior is somewhat being constrained and the search strategy guided by the user—whereby direct user involvement ultimately becomes relevant in terms of defining the satisficing behavior of the developer.

Adjusting aspiration levels

Direct user involvement during Sprints had a strong influence on aspiration levels. Simon [21] mentions aspiration levels as the *stop rule* signifying that a satisfactory alternative has been found and that the search strategy can be terminated. In other words, our aspiration levels act as criteria for whether or not an acceptable solution has been discovered.

On most occasions, a specific user interface was developed to meet the aspiration levels of the developer; and when evaluated by the user, the aspiration levels were either proven to be congruent with user's or not. Either resulting in acceptance of the proposed solution or a need for further improvement. The following passage, for example, reflects how the initial aspiration level for the user interface was adjusted after evaluating with User 1 in Episode 3.

> Besides that, he [User 1] has inputs for minor corrections to the user interface; for example formatting the dates, adding icons to finished requisitions, and removing social security numbers from the list overview [of requisitions in the application].

In this case, User 1 expresses a need for further improvement since the user interface does not provide the necessary information to support the work practice of bioanalysts at the hospital. If we view the overall design strategy chosen by the developer to be of a satisficing kind, direct user involvement in this example help fills in the specific details. Thus, we can better rely on abstracting information and working in simplified spaces, since the details approximating the real-world is introduced by direct user involvement.

Another point of observation to be made in this regard is that we find the direct user involvement in part to be guided by the principle of "I'll Know It When I See It" [2]. The following passage shows an example of this displayed after evaluating with User 1 in Episode 3.

> Even though he [User 1] was thrilled about the color label, it hit him today that it isn't important to see what the blood is tested for. It is, rather, very important to know how big the test tube should be.

Here, User 1 realizes that the initial revision to the user interface proposed in Episode 1 still was not the seemingly best solution. And we find this especially interesting since the user interface had already been revised during Episode 1 to become, at that moment in time, a satisficing solution for the user. What we gather from this, is that developers can ultimately only hope to design a satisfactory system by responding to input from the users. However, this is somewhat complicated by the users not knowing what they want until they see something similar to their needs—or realize that the current system is lacking a particular feature not previously apparent. Further, that because the users' aspiration levels for the intended system may very well change during development, so too must the developers' aspiration levels of the intended system change in response.

Thus, direct user involvement becomes relevant for adjusting the aspiration levels of developers, which is a necessity if we are to arrive at Design decisions ultimately resulting in a useful and usable product as experienced by the user.

Knowledge transfer

Tacit knowledge that is hard to put into words resides in all of us. And, of course, this poses a challenge when developing and designing software—since it implies that users may not be able to articulate what they want or need. So to exploit or benefit from this tacit knowledge, one must find a way to externalize it; e.g. via social interaction [17].

We find that knowledge transfer was very relevant during development and that the flow of tacit knowledge from user to developer strongly influenced Design decisions. In the following passage, for example, we see that the decision to implement a specific database design comes as a direct result of tacit knowledge about work practice being transferred during social interaction with User 1.

> After further consideration it is an obviously more correct way to model it because, in practice, it is not like you [the bioanalyst] are no longer responsible for the department associated with your [assigned] Team just because you offer assistance [to other bioanalysts].

In this case, the direct user involvement in Sprints had a significant impact on Design decisions; and was a deciding factor when choosing between different architectural courses of action. Another example of knowledge transfer influencing Design decisions can be seen in the following extract that describes the reasoning behind developing the logic for automatically assigning blood sample requisitions to bioanalysts working on a given day.

> First of all, the bioanalysts on a given Team do not necessarily come into work at the same time; not even though they are all scheduled to take blood samples during the morning round. And sometimes they do not come in at all (e.g. if they are sick). In other words, it is not sustainable to assign requisitions to the bioanalysts in advance.

Had the transfer of knowledge from user to the developer not been possible, the outcome would most likely not only be different—but also wrong to the point where the solution would not support the daily work practice of the bioanalysts. However, not all examples of knowledge transfer had a positive influence on Design decisions. In the following passage, we see how problems arose because certain social constructs contributed to cognitive challenges of the developer.

> The bioanalysts are aware that they are part of a Team—not least because it dictates which departments to service. But in practice, they do not consistently distinguish between being on a [specific] Team or at a [specific] department [...] when they speak colloquially of providing assistance, they refer to assisting a specific department—and not a specific Team.

This seemingly minor misconception proved to have great influence on Design decisions. Acting on the knowledge derived and generated through direct user involvement, the architectural modeling of the problem domain was done in a certain way—and when it was realized that this was in fact not the correct solution, larger parts of the application had to be refactored.

However, as an objection to the fault in this thinking, one can raise the point that without direct user involvement in each Sprint, the error may not have been found until much later—

which would have increased the costs of correcting it. And while this may be true, we find that it is a good example of a challenge related to direct user involvement and influence on Design decisions. This seems especially so in this case because the intended system in some way or other needed to reflect not only psychical objects (such as a department) but also social constructs (such as being part of a Team).

Design process

One of the core values of agile methods is the idea of developing rapid value through iterative software development. And in this sense, the agile development process facilitated through methods like Scrum is intended to be an iterative Design process, where requirements are generated, validated, and refined continuously [5].

Direct user involvement played an important role in validating and generating requirements throughout the Design process. Even though Product Owner 1 and Product Owner 2 was responsible for prioritizing User Stories on the Backlog during Episode 1, it was decided—and welcomed—that the intended users of the application should be an equal part in defining requirements; seeing as the application should, in the end, be beneficial to them and their work practice.

Not surprising, a lot of the requirements generated through direct user involvement in the Design process revolved around the user interface, since this is tangible and visible when evaluating a prototype. In the following, we see how specific requirements regarding available information is discovered.

> It is of particular importance that the user interface offers the possibility of seeing notes from the requester of the blood sample; e.g. special needs regarding measurements of fluids or the like. Further, it is important that a requisition has the date and time for when it has been ordered and completed [by the bioanalysts].

These requirements were not initially though off during Episode 1 and when developing User Stories—and were brought to attention after evaluating the prototype with User 1 in Episode 2.

However, not all requirements only had an impact on the user interface during the Design process. Other requirements generated through direct user involvement had an influence on the technical implementation. In the following extract, we see that interactions with User 1 provided further insights into technical requirements not discussed or thought off during Episode 1.

> There should not be any direct association between a bioanalyst and department without a Team [...] a bioanalyst is always assigned a Team on rounds. I some cases, a Team can consist of only one bioanalyst, but that does not change the logic of assigning requisitions [to bioanalysts]. In a case such as that, any given Team is still responsible for the same departments [as usual].

After meeting with User 1 in Episode 3 to clarify the work practice of the bioanalysts, specific requirements regarding the functionality of assigning blood sample requisitions were discovered and specified. Another example of generated requirements that had a direct impact on the technical implementation can be seen in the following passage.

> It turns out that the departments have to be finished in a certain order; and that this order is defined by doctors at the hospital. In other words, the application has to assign requisitions after what status, if you like, a given department has. In the sense that some departments are more important than others.

When evaluating the functionality of assigning blood sample requisitions during Episode 4, User 1 explains how certain departments at the hospital have to be finished first—because they have the most critical patients. And so, this, in turn, specifies very certain requirements regarding the technical implementation. If these requirements are not met, not only would the application not support the current work practice of bioanalysts at the hospital; it would also go against hospital regulation, and prove to be potentially dangerous for patients.

As evidenced, direct user involvement during Sprints had a significant impact on requirements; not only pertaining to the user interface but also in discovering and specifying requirements for the technical implementation. Also, the users evaluating the prototype during Episode 5 all express, that they perceive the direct user involvement to have had a clear impact on arriving at a solution supporting their work practice. Take, for example, User 3 who says the following when asked whether a good solution could be developed without interacting with end-users.

> I do not think so [...] It is easy to think that something is smart—but it has to be used. So I think that... We are the ones who need to use it.

Along the same lines, User 4 expresses that there is often a mismatch between the requirements of the customer—in this case, the directors or managers at the hospital—and the user. When asked the same question, User 4 replied the following.

> You have to talk with the users [of the application]. They [the directors] do not always know what we do. They are not the ones on the floor.

In summary, we find that direct user involvement during Sprints was very relevant for the generation of requirements during the Design process; and consequently for arriving at a solution that was perceived to be useful and supporting of the work practice at the hospital. Had we received all information from Product Owner 1 and Product Owner 2, it is likely that the perceived usefulness of the application would be lower—since the Design process and final product was highly influenced by the direct user involvement during Sprints.

Contingencies

During analysis, it became an apparent theme that contingencies in the development process could be related to complexity and uncertainty as presented in [14]. Here, a given development situation is characterized by having varying degrees of complexity and uncertainty—and developers must choose the appropriate approach dependent on what characterizes their situation. Complexity, in this case, represents the amount and diversity of relevant information available; and uncertainty represents availability and reliability of relevant information.

In the following, we will present findings on complexity and uncertainty as contingencies during the development process—and how direct user involvement in Sprints influenced these contingency factors.

User involvement reducing uncertainty

One way of generating information, and thereby reducing uncertainty, during a development process is to use prototypes [14]. This was very much the case in this study, since most of the information generated as a consequence of direct user involvement, was by way of evaluating prototypes. We find that a recurring theme during development was the generation of uncertainty during Sprints; uncertainties that could be reduced by evaluating with users. In the following passage, we see reflections in the diary commenting on uncertainties generated during Sprint 1.

> [...] the development of the user interface has moved from being complex because of [not knowing] how to build the user interface in Android, to being characterized by uncertainties concerning whether or not what I [the developer] has built, and if it is correct.

During the first Sprint, the development of the user interface generated uncertainty—lack of relevant information—about whether the solution would fit the users or not. We can see another example of Sprints generating uncertainty related to the user interface in the following extract.

> And I [the developer] am therefore not sure whether I have shown the most important information or not—or whether the user interface is at all intuitive.

In this case, further implementation of a different user interface during Sprint 3 generated uncertainty with regards to the appreciation of users. However, Sprints also generated uncertainty not related to the user interface. In the next passage, there is expressed concerns regarding how bioanalysts move around the hospital during shifts.

> There could be multiple reasons for a Team to always be at the same floor, and I [the developer] can, by the nature of things, only find this out by asking at the hospital.

This passage from Sprint 3 is an example of uncertainty generated during Sprints that have nothing to do with the user interface but are directly related to work practice and implementation of required functionality. A final example showing uncertainty produced during Sprints is the following extract questioning whether the functionality of the application supports the bioanalysts' work practice or not.

> It is particularly important that I [the developer] receive confirmation as to whether my implementation [...] is correct since this functionality is very sought after [by the bioanalysts].

In this example from Sprint 4, there is not much uncertainty regarding the structuring of the user interface—however, there are doubts as to the functionality; and how this supports the bioanalysts. Consequently, when evaluating with User 1, uncertainties generated in Sprints were reduced and clarified after evaluation. In the following passage, we see how uncertainty from Sprint 1 was reduced after meeting with User 1 in Episode 2.

> The meeting with him [User 1] was therefore ideal in relation to the uncertainty of the user interface. Moreover getting confirmation as to whether the structuring of the user interface was a good idea, and what information is essential for the work practice of a bioanalyst.

And the same goes for uncertainties related to work practice. Take the following passage, showing how User 1 helped clarify in what order bioanalysts move between the different departments at the hospital.

> I [the developer] asked if there was any structure as to how a Team moves between departments [...] And there was. I've mapped with him [User 1] which departments to finish in what order, and this is essential for how to assign requisitions [in the application].

During Episode 3, the order of departments to service first during a normal shift was clarified, which, in turn, is very important if the application was to support the work practice of bioanalysts. Seeing uncertainty as a contingency factor during the development process, we find that much of the uncertainty generated in Sprints could be reduced by user evaluation and direct user involvement. Not only about the user interface, but also work practice and functionality.

User involvement adding complexity

Complexity during the development process can be reduced through specification, abstraction and decomposition [14]. In this study, complexity was typically introduced by user involvement because the information generated by evaluating with User 1 needed to be structured and abstracted. This was especially so when information about work practice was generated through evaluation. In the following passage, we see how clarification of work practice during Episode 3 introduces complexity to the application.

> The application needs to be forthcoming with regards to the wish of assisting another department. Which means, in practice, that the bioanalyst wishing to assist needs to be associated with a different Team at that moment they wish to help with another department's unfinished requisitions.

In this instance, it becomes clear that the social construct of a Team at the hospital is somewhat fluid—and that the application needs to accommodate bioanalysts being part of multiple Teams at different points in time. Another example of complexity being introduced by direct user involvement can be seen in the following extract.

> [...] they [the bioanalysts] need to be assigned an equal number of requisitions for each department, starting with the most critical departments first. Where I [the developer] initially thought of the requisitions [to be assigned in the application] as being part of an overall pool [of requisitions]—in reality, it is more correct to think of

them [the requisitions] as being a cake to be split evenly between bioanalysts.

Here, we see that Episode 4 provides clarification as to how to assign requisitions correctly—but it also introduces complexity to the development process, because it turned out to be more complicated than first assumed.

During the development process, we find that developing artifacts was especially relevant in terms of reducing the complexity arising throughout the development process. One example is the following passage, explaining how modeling the relationships between entities helped structure the information generated by meeting with User 1 in Episode 3.

> Under all circumstances it is a prerequisite [...] that a Team is represented in both the application and the database. And seeing as though the application is already becoming complex [...] it was to me [the developer] the most logical step to model the application by [...] class diagram.

To manage the information generated by direct user involvement—and rising complexity in the application—modeling of the application proved useful. This can also be related to the intrinsic relation between complexity and uncertainty presented in [14]. In this sense, the countermeasure to complexity introduced by direct user involvement and prototyping is specification and modeling of the problem domain. Viewing complexity as another contingency factor, we find that complexity was added by direct user involvement throughout the development process.

However, an important observation is that the complexity added by user involvement did not necessarily have a negative impact on development or final product. On the contrary, the previous passages show how direct user involvement by adding complexity helped develop a more correct and useful solution. It is, though, evident that direct user involvement to a certain extent adds complexity to the development process; and that developers need to handle this effectively.

DISCUSSION

In the following section, we discuss our findings on Design decisions, Design process, and contingencies in the development process.

Design decisions

The findings reveal some interesting insights as to what direct user involvement is relevant for in relation to Design decisions. We find that direct user involvement during Sprints was relevant not only to decisions about the user interface—but also Design decisions regarding the technical implementation. This might be especially so in our study because the application was built as a custom product to specifically support the work practice of bioanalysts at the hospital. If the application was not meant to support the specific requirements of a well-defined user-group, it is possible that direct user involvement during Sprints would be more relevant to Design decisions about the user interface than functionality.

Further, it is clear that there is a big overlap between the satisficing behavior and aspiration levels of the developer. This was to be expected since Simon [21] presents them as both being integral to an actor's decision-making. However, what is also worth noting, is that the knowledge transfer from user to developer also influences the satisficing behavior and aspiration levels. The more knowledge about the user's needs, work practice, and behavior—the more our aspiration levels should eventually be aligned with the user's.

It is also evident from our findings that user feedback to a certain extent could be characterized by the principle of "I'll Know It When I See It", and that the aspiration levels of users may evolve. In this sense, direct user involvement is very relevant for developers if they are to reach a solution appreciated by users because direct user involvement helps to adjust the aspiration levels of those making Design decisions. If we only ever develop to our aspiration levels—without making sure that these are aligned with our users—we can not hope to always arrive at useful and usable products.

What we have not investigated or demonstrated in this study, is whether the aspiration levels of users continue to rise and evolve. It could be especially interesting to see if, and how, aspiration levels of users evolve over longer periods of time—and how the direct involvement of multiple users influence the satisficing behavior and aspiration levels of the developers.

Design process

Direct user involvement in Sprints was particularly relevant in the Design process, where it was required to understand and support the work practice of bioanalysts. We gather that this might be especially so in this case, because the intended application very much needed to support social constructs at the hospital; which in many ways reside as tacit knowledge in the bioanalysts. Therefore, social interaction with users helped generate requirements to both the user interface and functionality.

It is also apparent from our findings that the users evaluating the prototype after the final Sprint perceived it to be useful and supporting of their work practice. And that they saw the role of direct user involvement in the Design process as being a key part to this. This is relevant to the Design process for several reasons. The most obvious one being that final product is usable, and found useful by users; which should be the end-goal of the Design process. But also, direct user involvement in Sprints means that users feel their needs are being met and that they are being heard throughout the Design process. We find, that this is a very important benefit to direct user involvement in Sprints if we are to meet the criticism of agile methods neglecting the difference between the customer and the user.

Contingencies

What we gather from our findings of contingencies during the development process is primarily three observations. The first observation is that is evident, that uncertainty is generated throughout the development process as the application progresses. This also showcases the importance of direct user involvement, since uncertainty was reduced or eliminated by way of direct user involvement and evaluation of prototypes. Had direct user involvement not been possible, much of the uncertainty generated in Sprints would be unresolved. An argument against this would be that other stakeholders could in practice be conferred with—an example being other managers or directors from the hospital. While this is true, it could also be made the case that, as expressed by User 4, the customers do not necessarily know specific work practices of the intended users.

The second observation is that the presence of uncertainty during the development process is closely related to satisficing behavior and aspiration levels. It becomes clear, that when faced with different uncertainties, pertaining to various aspects, during the development process—one can only devise a solution best meeting the aspiration levels set by one-self. If we are in a position where we do not know what the user want, we can only hope to develop a solution that *we* find satisficing; and as exemplified in this study, the aspiration levels of the developer is not always the same as the user's. In this regard, it becomes essential—in developing a useful solution for the user—that uncertainties generated in Sprints are reduced. And one, in our case, effective way of doing this, was by having direct user involvement in Sprints.

Lastly, the third observation is that, although we find direct user involvement introduced complexity throughout the development process, there are some limitations inherent to this study that might otherwise provide some more insights. The most important one is that we only involved User 1 throughout the development process. This means that there is only one user that introduces different requirements, ideas, and input. If there were multiple users directly involved in each Sprint, the amount of information to structure and understand would most certainly by bigger. Therefore, the complexity introduced by direct user involvement would, in that case, likely be even greater. This would also pose a challenge in evaluating what input to value and prioritize more than others. In our study, we only needed to appreciate the needs of a single user—but in the case of multiple users directly involved in each Sprint, it would require more attention towards abstraction and specification of expressed requirements, needs, and opinions.

CONCLUSION

This paper presented an exploratory study investigating what direct user involvement is relevant for in relation to Design decisions and Design process—and to what extent direct user involvement influence the development process' contingencies. The exploratory study presents findings from a diary kept while developing an Android application for support the work practice of bioanalysts at the local university hospital.

We find that direct user involvement was relevant for several things related to Design decisions. Evaluation with users helped define the satisficing behavior of the developer; proved relevant in adjusting aspiration levels of the developer; and was valuable in transferring tacit knowledge from user to developer. Regarding the Design process, we find that direct user involvement has been particularly relevant for generation and validation of requirements for the application.

Lastly, we find that direct user involvement influenced both complexity and uncertainty as contingencies during the development process. Uncertainty was generated during Sprints throughout the development process, and user involvement helped reduce uncertainties by evaluating prototypes and social interaction. In turn, direct user involvement added complexity to the development process that developers must handle appropriately.

Limitations

We acknowledge the limitations inherent to this exploratory study. Since this is a case study investigating the development of a custom product, findings and circumstances would likely be different in other organizations and development processes.

The development process, although mimicking Scrum, was performed with only one developer and one user. Had the development team been larger—and had more users been involved in Sprints—the findings of the study could be different. However, the study involved different stakeholders consuming different roles as in a regular Scrum process, and multiple users evaluated the prototype after development was concluded. Further, the development process was limited to only three months, which could have an impact on data collection and findings. We have tried to negate this by choosing diaries as the data collection method since this provides rich and contextual data.

Future work

Additional studies should be performed to understand how direct user involvement influence Design decisions and Design process in larger teams—and over longer periods of time. This could also provide an opportunity of investigating how the direct involvement of several users in Sprints influence Design decisions and Design process.

Lastly, there is lacking empirical studies investigating how contingencies in the development process can be used for effective user involvement strategies. We have shown that direct user involvement influence both complexity and uncertainty, and additional studies could build on this empirical evidence; suggesting how these could be used to decide on relevant user involvement in Sprints.

ACKNOWLEDGMENTS

We would like to thank all participants from Clinical Biochemistry at Aalborg University Hospital for their contribution to this exploratory study, and their role in the development process. Specifically, we would like to thank Lars Domino, Tina Beith, and Claus Gyrup Nielsen for playing a big role during this study.

REFERENCES

1. Stefan Blomkvist. 2005. Towards a Model for Bridging Agile Development and User-Centered Design. In *Human-Centered Software Engineering — Integrating Usability in the Software Development Lifecycle*, Ahmed Seffah, Jan Gulliksen, and Michel C. Desmarais (Eds.). Number 8 in Human-Computer Interaction Series. Springer Netherlands, 219–244.
2. Barry Boehm. 2000. Requirements that handle IKIWISI, COTS, and rapid change. *Computer* 33, 7 (2000), 99–102.

3. 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 (2015), 163–181. DOI: http://dx.doi.org/10.1016/j.infsof.2015.01.004

4. Alan Bryman. 2008. *Social Research Methods*. Oxford University Press.

5. Alistair Cockburn and Jim Highsmith. 2001. Agile software development, the people factor. *Computer* 34, 11 (2001). DOI:http://dx.doi.org/10.1109/2.963450

6. Jim Highsmith and Alistair Cockburn. 2001. Agile software development: the business of innovation. *Computer* 34, 9 (Sept. 2001), 120–127. DOI: http://dx.doi.org/10.1109/2.947100

7. Leif Obel Jepsen, Lars Mathiassen, and Peter Axel Nielsen. 1989. Back to thinking mode: diaries for the management of information systems development projects. *Behaviour & Information Technology* 8, 3 (1989), 207–217. DOI: http://dx.doi.org/10.1080/01449298908914552

8. Timo Jokela. 2004. When good things happen to bad products: where are the benefits of usability in the consumer appliance market? *Interactions* 11, 6 (2004), 28–35.

9. Elaine Kant. 1985. Understanding and automating algorithm design. *IEEE Transactions on Software Engineering* 11 (1985), 1361–1374.

10. Kati Kuusinen. 2016. BoB: A Framework for Organizing Within-Iteration UX Work in Agile Development. In *Integrating User-Centred Design in Agile Development*, Gilbert Cockton, Marta Lárusdóttir, Peggy Gregory, and Åsa Cajander (Eds.). Springer International Publishing, Cham, 205–224. http://link.springer.com/10.1007/978-3-319-32165-3_9 DOI: 10.1007/978-3-319-32165-3_9.

11. Kati Kuusinen, Tommi Mikkonen, and Santtu Pakarinen. 2012. Agile User Experience Development in a Large Software Organization: Good Expertise but Limited Impact. In *Human-Centered Software Engineering*, Marco Winckler, Peter Forbrig, and Regina Bernhaupt (Eds.). Vol. 7623. Springer Berlin Heidelberg, Berlin, Heidelberg.

12. Marta Larusdottir, Jan Gulliksen, and Åsa Cajander. 2016. A license to kill – Improving UCSD in Agile development. *Journal of Systems and Software* (2016). DOI:http://dx.doi.org/10.1016/j.jss.2016.01.024

13. Jonathan Lazar, Jinjuan Heidi Feng, and Harry Hochheiser. 2010. *Research methods in human-computer interaction*. John Wiley & Sons.

14. Lars Mathiassen and Jan Stage. 1992. The Principle of Limited Reduction. *Information Technology and People* 6 (1992), 171–185.

15. Deborah J. Mayhew and Marilyn M. Tremaine. 2005. A basic framework. *Cost-justifying usability: An update for the internet age* (2005), 41–101.

16. Peter Naur. 1972. An experiment on program development. *BIT Numerical Mathematics* 12, 3 (Sept. 1972), 347–365. DOI: http://dx.doi.org/10.1007/BF01932307

17. Ikujiro Nonaka and Hirotaka Takeuchi. 1995. *The knowledge-creating company: How Japanese companies create the dynamics of innovation*. Oxford university press.

18. Johnny Saldaña. 2015. *The coding manual for qualitative researchers*. Sage.

19. Ken Schwaber and Jeff Sutherland. 2011. The scrum guide. *Scrum Alliance* 21 (2011).

20. Herbert A. Simon. 1955. A Behavioral Model of Rational Choice. *The Quarterly Journal of Economics* 69, 1 (1955), 99–118.

21. Herbert A Simon. 1972. Theories of bounded rationality. *Decision and organization* 1, 1 (1972), 161–176.

22. Desirée Sy. 2007. Adapting Usability Investigations for Agile User-centered Design. *J. Usability Studies* 2, 3 (2007), 112–132.

23. Antony Tang and Hans van Vliet. 2015. Software Designers Satisfice. In *Software Architecture*, Danny Weyns, Raffaela Mirandola, and Ivica Crnkovic (Eds.). Vol. 9278. Springer International Publishing, Cham, 105–120. DOI: http://dx.doi.org/10.1007/978-3-319-23727-5_9

24. Carmen Zannier, Mike Chiasson, and Frank Maurer. 2007. A model of design decision making based on empirical results of interviews with software designers. *Information and Software Technology* 49, 6 (June 2007), 637–653. DOI: http://dx.doi.org/10.1016/j.infsof.2007.02.010

25. Tina Øvad and Lars Bo Larsen. 2015. The Prevalence of UX Design in Agile Development Processes in Industry. In *Agile Conference (AGILE), 2015*. IEEE, 40–49. DOI: http://dx.doi.org/10.1109/Agile.2015.13