Tampere University

SAMPO SUONSYRJÄ

Data-Driven Software Development with User-Interaction Data

Tampere University Dissertations 84

Tampere University Dissertations 84

SAMPO SUONSYRJÄ

Data-Driven Software Development with User-Interaction Data

ACADEMIC DISSERTATION
To be presented, with the permission of
the Faculty of Information Technology and Communication Sciences
of Tampere University,
for public discussion in the auditorium S2
of the Sähkötalo, Korkeakoulunkatu 3, Tampere,
on 26 June 2019, at 12 o'clock.

ACADEMIC DISSERTATION

Tampere University, Faculty of Information Technology and Communication Sciences
Finland

ISBN 978-952-03-1144-5 (print)
ISBN 978-952-03-1145-2 (pdf)
ISSN 2489-9860 (print)
ISSN 2490-0028 (pdf)
http://urn.fi/URN:ISBN:978-952-03-1145-2

PunaMusta Oy – Yliopistopaino
Tampere 2019

Abstract

Suonsyrjä, Sampo

Data-Driven Software Development with User-Interaction Data

Gathering feedback has always played an important role in product design. For software development, user-centered design of systems has been a trend already since the 1980's. Similarly, making decisions based on quantitative data is on the rise. Software-intensive companies, such as Facebook and Google, already collect and analyze loads of data about their users – especially for marketing purposes.

However, using user-related data in data-driven software development is still in its infancy. Given the increasing speed of software development and the need for ever-tightening user engagement, new solutions for faster feedback mechanisms are clearly needed. Thus, the research problem of this thesis was to produce an actionable set of tools and methods for using user-interaction (U-I) data in software development.

To solve this, we used Design Science and Action Design Research strategies. A set of tools and methods for using U-I data were designed and evaluated. For these, we conducted exploratory, explanatory, and improvement case studies with three software teams from different organizations. Additionally, we surveyed a larger set of software practitioners.

The results are threefold. Firstly, our tools assist practitioners in the collecting of U-I data technically. We identified five U-I data collecting techniques, designed a framework for their selection, developed an open source collecting tool, and designed a demonstrative tool stack to cover analytics end-to-end. Secondly, the thesis presents results for how to use U-I data in software development. Four analysis and four use objectives for U-I data were found. In addition, the designed U-I data utilization method presents a three step guide for how to start the use of U-I data. Thirdly, the synthesis of the U-I data objectives with the objectives of iterative software development cycles highlights several opportunities of using U-I data on a methodological level. To understand the practical level, the results also describe a set of challenges of using U-I data.

The thesis research contributes in developing the fast feedback mechanism of gathering quantitative data from how users use software systems. The high velocity of collecting feedback is essential for software-intensive organizations enabling the data-driven software development. On a practical level, many of the tools designed during this thesis have been integrated into the software systems of practitioners. Moreover, the use of U-I data is now easier for software teams because the results of this thesis explain its opportunities and challenges. As a whole, the thesis provides software teams with an actionable set of tools and methods that assists them in responding to their users' needs faster than before.

Keywords: software engineering, software analytics, data-driven software development, post-deployment data, user-interaction data

Preface

"The truth is on the top of a mountain, but we all see it from our own height."
– Engraved in a tomb stone of a loved one.

The cited philosophy above reminds us to check up on how others might have different views of the world, even if the thing might seem obvious to us. In this thesis, we designed new analytics tools for people to create additional views to the world. I feel it is essential to understand that these tools provide you with one view to that mountain top - a view that can be important yet seldom complete on its own.

For a researcher, the enthusiasm to go around and find out not just one, but plenty of views to a topic is crucial. In a research effort as exhausting as a phd thesis however, the plain enthusiasm of a single doctoral student won't carry you the whole way. Fortunately, I have been more than blessed to have had a group of the most smart, skillful and loving people around me. With you, I see we have been able to conquer at least one of the peaks of the world.

First of all, I want to thank my two supervisors, prof Kari Systä and prof Tommi Mikkonen. Our sessions on the whiteboard and your advice during the thesis work have left me with fond memories and in great appreciation. I trust and value you, but I have also felt trusted and valued by you. In such an environment it has been easy to come up with ideas, to share them and believe in them, and finally to make them happen.

I am thankful for prof Ville Leppänen and prof Jürgen Münch for agreeing to be the pre-examiners of the thesis and for prof Jan Bosch for acting as the opponent.

I have had the most fulfilling time of my working life in doing the thesis work. What made it so special, was the amount of how much I've learned from the wise group of colleagues I got to work with. So I want to thank Dr. Terhi Kilamo, Dr. Kati Kuusinen, and Dr. Outi Sievi-Korte for all of the research knowledge and hard work they were willing to selflessly exercise on helping me. Similarly, I'm grateful for finding a group of fellow doctoral students who shared the same enthusiasm for software startups that I have. However, I want to thank Dr. Laura Hokkanen and Henri Terho not only for the work on the exciting research domain but also for the shared tears of joy and sorrow in the life of a startup doctor(al student). With you, we let the passion be seen also concretely in our startup company, Taplia. This was an exceptional learning place for me and I got to study how continuous delivery works in practice. However with Esa Kaarna and Joni Hämäläinen, the delivery did not include only new software features, but continuous smiles and friendship as well.

One of the key enablers of the thesis work was the forefront research project Need for Speed. There are not many similar opportunities where a novice researcher can make connections with some of the leading software professionals in Finland. Similarly, I am grateful for getting to work with the great teaching group in the laboratory of pervasive computing. I want to thank Timo Lehtonen, Esko Pekkarinen, Timo Aaltonen, Tero Ahtee, Samuel Lahtinen, Teemu Laukkarinen, Anna-Liisa Mattila, Mikko Nurminen, Ulla-Talvikki Virta, and all others who I crossed path with in our cases but who should remain unnamed due to research anonymity.

Finally, I want to thank my family who has always had my back no matter what. I am grateful to have such a wise mother, a loving grandma, and a supportive sister. This was not my first crazy ambitious project, and it probably is not the last one either. But without you, there would not have been even the first one.

Tampere 9.6.2019

Contents

Abbreviations, Terms and Definitions

A/B Testing	-	Experimentation with two variants of a software feature.
Add-on	-	An extra feature on top of the basics of an application.
Action Design Research	ADR	Research strategy combining Design Science and Action Research.
Agile methods	-	Software engineering methods introduced in the early 2000's valuing close customer collaboration.
Analytics	-	The gathering of data and their turning into insights.
Application Specific Integrated Circuit	ASIC	
Aspect-Oriented Programming	AOP	Programming Paradigm
Application Programming Interface	API	
Artifact	-	Design Science and ADR design and evaluate IT focused artifacts, e.g. frameworks, methods and instantiations.
Business to Business	B2B	Business model where other businesses are customers.
Business to Consumer	B2C	Business model where consumers are customers.
Big Data	-	Large and complex data sets requiring advanced and unique technologies [1].
Build-Measure-Learn	BML	Three part cycle for development in Lean Startup.
Clickstream Data	-	"Information about the sequence of pages or the path viewed by users as they navigate a website"[2].
Continuous Deployment	CD	The practice of deploying new software versions as soon as their development is finished.
Customer Data	-	Data concerning the background and characteristics of a customer.
Data-Driven Software Development	-	Steering software development work with data.

Development & Operations	DevOps	Movement of tightening the collaboration between development and operations teams.
Design Science (Research Methodology)	DS & DSRM	Research strategy for building and evaluating IT artifacts.
End-Users	-	The actual intended users of a system, not testers, developers or managers who can still use the system.
Experiment-Driven Development	-	Guiding subsquent development activities with factual feedback from previous software versions [3].
Field-Programmable Gate Array	FPGA	
Goal/Question/Metric	GQM	An approach for setting metrics based on goals.
Graphical User Interface	GUI	
Human-Computer Interaction	HCI	
Highest Paid Person's Opinion	HiPPO	
Hypothesis Experiment Data-Driven Development	HYPEX	Model for experimentation in software development.
Instrumentation	-	Insertion of additional statements to source code.
Information Systems	IS	Research field
Information Technology	IT	
Interaction Data	-	Data covering the interactions of developers with software artifacts.
Iterative development	-	Development where new functionalities are build on top of the previous ones.
Key Performance Indicator	KPI	
Log Data	-	Output lines from tracing statements.
Machine Learning	-	The use of statistical techniques to make computers learn from data.
Mining Software Repositories	MSR	Research field for collecting and analyzing data about software systems and projects from software repositories [4].
Minimum Viable Product	MVP	Version of a product that is produced with minimum resources but that is still able to produce reliable results for validating a business/development hypothesis.
Need for Speed	N4S	Research program
Open-source	-	License that grants open rights.
Post-Deployment Data	PDD	Data collected after the deployment of a software system.
Practitioner	-	An employee in the software industry.
Production Environment	-	An environment where a system is used by its end-users.

Qualitative / Quantitative Customer-Driven Development	QCD	Method for experimentation in software development.
Research Question	RQ	
Software-as-a-Service	SaaS	Delivery model for centrally hosted software systems.
Software-Intensive	-	Organizations that have based their business model on software development.
Software Development	-	The development of software.
Software Engineering	-	The research field studying software.
Software Operation Knowledge	SOK	In-the-field knowledge of software performance, quality, usage, and end-user feedback.
Staging Environment	-	A testing environment similar to production but where the system is not used by the end-users.
Telemetry	-	Streaming Log Data
User-Centered Systems Design	UCSD	
User-Interaction Data	U-I Data	Data concerning the interactions of users with the system in the UI level.
User Interface	UI	
Unobtrusive	-	Not introducing changes to source code.
Usage Profiles	-	"Sequences of events that end-users execute on a GUI"[5].
User eXperience	UX	

List of Publications

The thesis consists of an introductory part and the following original publications:

I Suonsyrjä, S., Mikkonen, T. "Designing an Unobtrusive Analytics Framework for Monitoring Java Applications", *International Workshop on Software Measurement (IWSM)*, pp. 160−175 Jan. 2015.

II Suonsyrjä S., Systä K., Mikkonen T., Terho H. "Collecting Usage Data for Software Development: Selection Framework for Technological Approaches", in *In Proceedings of The Twenty-Eighth International Conference on Software Engineering and Knowledge Engineering (SEKE)* 2016.

III Suonsyrjä S., "Eeny, Meeny, Miny, Mo... A Multiple Case Study on Selecting a Technique for User-Interaction Data Collecting" in *International Conference on Agile Software Development (XP)* 2017. pp. 52-67.

IV Suonsyrjä S., Hokkanen L., Terho H., Systä K., Mikkonen T., "Post-Deployment Data: A Recipe for Satisfying Knowledge Needs in Software Development?" in *Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA)* 2016. pp. 139-147.

V Suonsyrjä S., Sievi-Korte O., Systä K., Kilamo T., Mikkonen T., "Objectives and Challenges of the Utilization of User-Interaction Data in Software Development", in *Proceedings of the 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* 2018. pp. 357-361. IEEE.

VI Terho H., Suonsyrjä S., Systä K., Mikkonen T., "Understanding the Relations Between Iterative Cycles in Software Engineering", in *Proceedings of the 50th Hawaii International Conference on System Sciences (HICSS)* 2017.

The publications are reproduced in accordance of the publication permission schemes of the publishers. The contributions and the role of the candidate in each publication is described in the following.

In Publication **I**, the candidate planned and analyzed the study with the second author who had a significant contribution in structuring the Publication and writing some of the sections. The candidate developed the demonstrative application needed for the study and wrote majority of the publication. The candidate is the lead author of the Publication.

In Publication **II**, the candidate planned, conducted, analyzed, and reported the study with the second author who assisted in the work. The candidate wrote the majority of the Publication and the rest of the authors contributed in writing some parts of the sections. The third author had a significant contribution in conducting the study with the candidate. The candidate is the lead author of the Publication.

In Publication **III**, the candidate planned, conducted, analyzed, and reported the study by himself. Two researchers commented on the writing. The candidate developed the related JavaScript tool for collecting user-interaction data, which is open-source and available in GitHub[1]. The candidate is the sole author of the Publication.

In Publication **IV**, the candidate planned, analyzed, and reported the study and wrote the majority of the Publication. The second author had a major role in conducting the study: She formed the questionnaire, sent it to the respondents, and collected the answers. She participated in analyzing the results and wrote one subsection of the Publication. The third author wrote some of the background section and the rest of the authors contributed in polishing the Publication and writing minor parts of it. The candidate is the lead author of the Publication.

In Publication **V**, the candidate planned, conducted, analyzed, and reported the study. The second and the third author assisted in the analysis and together with the fourth and fifth author they assisted in polishing the Publication. The candidate wrote the majority of it and the rest of the authors contributed on writing some smaller segments. The second, third and fifth authors participated in gathering the research data with the candidate. The candidate is the lead author of the Publication.

In Publication **VI**, the candidate is the second author. He planned, conducted, analyzed, and reported the study with the rest of the authors. Majority of the Publication was written equally by the first and the second author. The rest of the authors contributed on writing some smaller segments and polishing the Publication.

[1]https://github.com/ssuonsyrja/Usage-Data-Collector

1 Introduction

The wide-spread availability of Internet and high-speed network capabilities in general have enabled software-intensive companies of the 00's, such as Facebook, Google, and Netflix, to become parts of the daily lives of many of us. At the same time, the way these tech giants and an increasing number of other companies are delivering the software has changed from product business to services [6]. The change has been radical in that it has allowed software systems to become more adaptive than before and to personalize content for individual users with the help of automatically collected data. For example in an online booking service, tickets for the same flight can cost more for users categorized as business travelers than for vacationers, and while watching the same television show from a streaming service two neighbors might see different ads if one has been marked to have kids and the other has not [7]. In these examples, the personalization is based on customer data, i.e., data about the background and characteristics of a customer. Similarly, companies can gather data about the habits of groups of people, such as what they are buying or watching, and then recommend additional products or movies for individuals based on these wider habits. Such use of big data, i.e., large and complex data sets requiring advanced and unique technologies [1], characterizes the data-driven trend of decision-making of today's business environment.

In the field of software engineering, an even longer standing trend than the content personalization has been the user-centered design of software systems. The term User-centered systems design (UCSD) was coined by Norman and Draper already in 1986 [8]. Especially after the introduction of Agile methods in the early 2000's, users have been more involved in the development phase of software systems. The values in the Agile Manifesto [9] have guided software teams towards closer customer collaboration and responding to changes, in user requirements for example. The collaboration between customers or users and software developers is aimed to be close to ensure that enough information is shared among them, and for example Dybå and Dingsøyr [10] mention specifically customer collaboration as a benefit of agile development. However, the ways of getting feedback from users are often manual and thus highly laborious. For example, having individual email conversations with users or meeting people face to face takes a lot of time and effort. Typically, such feedback collecting mechanisms also produce unstructured qualitative information, which makes the feedback collecting and analyzing from larger crowds of users rather difficult. By no means are such information useless - rather they provide developers with highly important insights - but they also lack some of the benefits that we now have witnessed being used by the tech giants with the automatically collected customer data.

Over the last decade, new technologies for software development have emerged [11], and especially technologies for delivering new software versions to users are becoming extremely sophisticated. Nowadays, it is possible to have a new software release in use within seconds from being finished by its developer [12]. This extreme decrease in deployment times enables the shortening of the whole development cycle which again has multiple radical effects. On a practical level, the use of the different continuous practices has been increasing [13] and the collaboration between software development and operations is tightening. This movement has been summarized under the term DevOps (**Dev**elopment & **Op**erations), which was coined in 2009 as the first "DevOps Days" conference was held [14]. On a methodological level, there are effects as well. Although cyclical learning methods, such as the plan-do-check-act (often credited to Deming, e.g. [15]) or the scientific method itself, have been around for ages, their speed and scalability could now get to a new level. A key enabler for these learning mechanisms is the gathering of data and their turning into insights, i.e., analytics. However, the manual feedback collecting methods cannot keep up with the required level of velocity. Therefore, new rapid automated means of collecting feedback from the ever-changing software versions are required as well. With advances in this field, software engineering can get towards the continuous evolution of software functionalities although the topic is still in its infancy [13].

The encouraging examples above about using customer data for marketing and content personalization are inspiring, and similar things could work in the software development context as well. In this thesis, we focus on Post-Deployment Data (PDD) and especially on User-Interaction (U-I) data, i.e., data that is collected after the deployment of a software system and that concerns the interactions of users with the system in the user interface (UI) level. Although the trend of guiding software development work with data, i.e., data-driven development e.g. [16], has already been around for a while the increasing use of software development methods that value users and their feedback are now creating the need specifically for user-related data collecting and use. Enabling this, the advances in software deployment technologies and in network connectivity are forming a common infrastructure where we now have the possibility to not only collect data but also use the data for changing the software systems in rapid development cycles. However, it is specifically this requirement of the new level of speed in development that creates the need for more rapid − and thus automated − feedback collecting mechanisms. Figure 1.1 illustrates these trends and advances motivating and enabling this thesis research on data-driven software development with U-I data.

1.1 Research Goals and Questions

The main goal of the thesis is to produce an actionable set of tools and methods for software teams to increase their capabilities of moving towards data-driven software development with U-I data. More specifically, the work aims at studying, developing, and discussing tools, techniques, methods, and processes for using U-I data in software development. While the product of the main goal is focused on helping practitioners, the thesis work aims to contribute equally to the research field of software engineering. Thus, the aim and the target audience of the thesis is twofold. For academia, PDD and the related methods of software engineering are studied to examine the state of the art and to bring it further. For practitioners, we develop concrete methods and tools in close co-operation with software-intensive organizations joining us in DIMECC Need for Speed (N4S) program[1]. The two aims support each other, and we think that this practical

[1]http://n4s.dimecc.com/en/

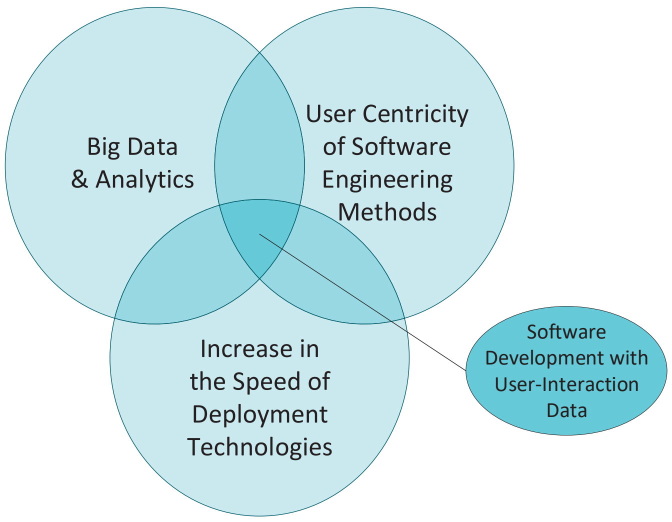

Big Data & Analytics
User Centricity of Software Engineering Methods
Increase in the Speed of Deployment Technologies
Software Development with User-Interaction Data

Figure 1.1: Technology trends motivating and enabling the thesis research on data-driven software development with user-interaction data

setting assists the academic contributions in both communicating about the research and for conducting it in vivo.

From the perspective of research areas, the aims of this software engineering research effort divide it into three. As technological research, the objective is to study techniques for the automatic collecting of software PDD. Then from more of a software development perspective, we examine U-I data, and objectives for their analysis and use. Finally from the point of software methodologies, we study the ways of utilizing PDD in software development processes. To clarify a research question (RQ) for each, the following points elaborate these three segments further.

RQ1. How to select a technique for collecting user-interaction data?

First of all, collecting U-I data is usually treated as an add-on, i.e., an extra feature on top of the basics of an application. Because of this, the original programming techniques, tools and libraries of an application are not sufficient and the collecting of U-I data needs additional insertions of such. The selection of these techniques is no easy task however, because even understanding what kind of options are available and what kind of characteristics they have takes a lot of effort.

RQ2. How to use user-interaction data in software development?

Secondly, we aim at gaining an understanding of the objectives software teams have for the utilization of U-I data. In addition, we strive to design guidance for teams to start the utilization of U-I data. The objectives are studied both from practical and theoretical standpoints. The range of objectives can be wide however, and therefore we aim to categorize the different analysis and use objectives software teams have for U-I data.

RQ3. What kind of opportunities and challenges are there in the utilization of user-interaction data?

Finally in the third research segment, we study the integration of U-I data into the processes of software teams. We approach this in two ways. Conceptually, we analyze the opportunities of using U-I data to complement different software development approaches. Practically, we study the challenges the software teams face in such integrations.

1.2 Research Methods and Context

The aim of the thesis is to produce new information technology (IT) focused artifacts in a context where practitioners do not have experience in using similar artifacts beforehand. Therefore, the research in the thesis requires research strategies that include intervening in the works of software teams. Popular constructive research strategies in the field of software engineering such as Design Science (DS) [17] and Action Design Research (ADR) [18] suit these aims and requirements and they are used also in this thesis work. Similarly, a mix of research methods is required to complement these strategies. For example, case studies make it possible for us to get down to practical contexts, which again assists in the evaluation of the artifacts. Related to the collecting of research data, methods such as surveys, interviews, and observation are used in the studies of the thesis.

Given the actionable and practical aim of the thesis, the context in which the thesis work is done becomes important. Even if the thesis work is published with open access rights throughout the world, its results and insights are likely to be diffused mainly near the grounds of its original studies. To get the best possible outcome for this most probable audience, the artifacts need to be designed and evaluated in contexts as close to theirs as possible. The thesis work is carried out in the Finnish N4S program[2]. In this program, some of the leading software-intensive companies in Finland have joined forces with Finnish research organizations to increase their software engineering related capabilities of delivering value rapidly to customers and to gather and generate knowledge on and around the subject. These organizations provide researchers and their studies with both an industrial context and an interested audience for the outcomes.

1.3 Scope and Contributions

The research in this thesis is divided into three segments. Together, these segments form a guidance that we aim at software teams going towards data-driven software development with U-I data. The first segment focuses on U-I data collecting techniques and their selection. The second segment examines objectives for such data and the utilization of U-I data. The third looks into integrating the data into software processes by examining the related opportunities and challenges. In this sense, the third segment builds on top of the first two. Together, they form the contribution of the thesis research as illustrated in Figure 1.2.

The research in the first segment concentrates on U-I data collecting techniques and it includes both development and evaluation of different techniques and tools. In addition, we design a tool stack that technically covers the utilization of U-I data from the collecting to the visualization of the data. The contributions of the first segment are listed as follows:

- Identification and evaluation of five U-I data collecting techniques.

[2]http://n4s.dimecc.com/en/

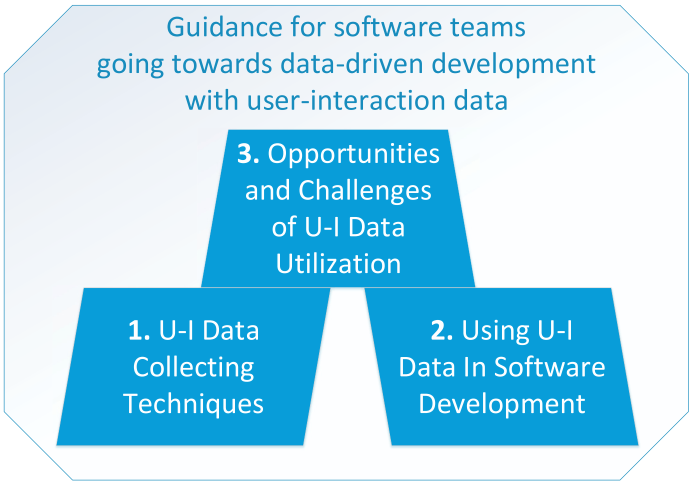

Guidance for software teams going towards data-driven development with user-interaction data
3. Opportunities and Challenges of U-I Data Utilization
1. U-I Data Collecting Techniques
2. Using U-I Data In Software Development

Figure 1.2: Three research segments forming the main contribution of the thesis: A guidance for software teams going towards data-driven development with user-interaction data.

- Selection method for U-I data collecting techniques.
- Open source tool for collecting U-I data from JavaScript applications.
- Design of a demonstrative tool stack for unobtrusive analytics.

The second segment focuses on the use of U-I data. Firstly, we examine the objectives teams have for U-I data. Secondly, we study the steps they take as they start the use of such data and model those steps by designing a method that other software teams can also use as a guidance. The contributions of the second segment are the following:

- Categorizations of U-I data types and objectives for analysis and utilization.
- Utilization method for U-I data.

The third segment builds on the research work done in the first two segments. We study how software teams integrate the use of U-I in their work by analyzing the use opportunities and by examining software teams about the challenges they face. The summarized contributions of the third segment are as follows:

- A synthesis of U-I data objectives and targets of iterative SW development methods.
- Categorization of the challenges in starting the use of U-I data for SW development.

From the beginning of the research work towards its end the stress on data collecting techniques shifts to the use of data, and then to the associated methods and processes. The main focus of the research is on automatically collected PDD and more specifically, on U-I data. Such data forms from the actions of users, and so we are especially interested in quantitative data that is generated after the deployment to environments where human users are interacting with the software system. This scopes out, for example, data that is generated by automated testing, hardware related performance data, and data that is gathered in qualitative format such as feedback surveys.

The research on U-I data utilization in software processes is scoped by having the software teams as the users of the data. This limits the participants of the studies to the software team members, excluding personnel for example from marketing and business development. However, even if our focus is on data use objectives related to software engineering, other views are not entirely scoped out. The software teams might have responsibilities in such organizational functions themselves, and in such cases the objectives are included. On the other hand, the thesis scope excludes research on legal matters related to U-I data. We want to point out that these are important to review when data is collected from user activities, but since such are country specific and require periodic updating they are beyond the scope of the research. In addition, data analysis methods such as machine learning algorithms are utilized during the research work, but they themselves are not under research in this thesis.

1.4 Structure of the Thesis

The rest of the introductory part to this thesis is divided into four chapters and they are structured as follows. Chapter 2 presents the background of the thesis work covering the fields of software analytics, PDD, and the key approaches for data-driven software development. Additionally, we take a look at how the scientific literature has examined the challenges in using user related data-driven approaches. Chapter 3 describes the research design including the strategies and methods used in this doctoral research work along with an introduction to the software teams who participated in the research. We also consider the validity of the research design. In Chapter 4, the results are first examined per research question and then summarized per publication. Chapter 5 revisits the research questions and discusses the contributions of the thesis. In addition, we analyze the trustworthiness of the designed artifacts, and present implications for future work. Finally, Chapter 6 draws the conclusions of the thesis.

After the introductory part, the six original research papers are reprinted in their original format.

2 Background and Related Work

In this chapter, we first describe the general concept of software analytics and take a look at the related software data in Section 2.1. In Section 2.2, we continue by narrowing down to PDD and by describing what kind of knowledge needs in software engineering have been identified by the scientific literature. In Section 2.3, we then cover topics related to our work on the field of data-driven software development: methods for collecting user related data, its analysis, and experimentation systems that utilize that data. Finally in Section 2.4, we finish the chapter with a look into the identified challenges of using such data in software development.

2.1 Software Analytics and Software Data

All over the world, products and services are being digitalized in growing numbers. Simultaneously, these software rich goods are producing data in quantities unlike ever before. To make sense of the information shockwave, analytical means of turning the vast data sets into actionable insights are required. This is where analytics comes in. For example, Davenport and Harris [20] define the concept as "extensive use of data, statistical and quantitative analysis, explanatory and predictive models, and fact-based management to drive decisions and actions." Similar to [20], also Kaushik [19] sees the analytics concept to include different analyses and models as a way to turn data into insights that support decision making. Figure 2.1 illustrates how analytics builds from

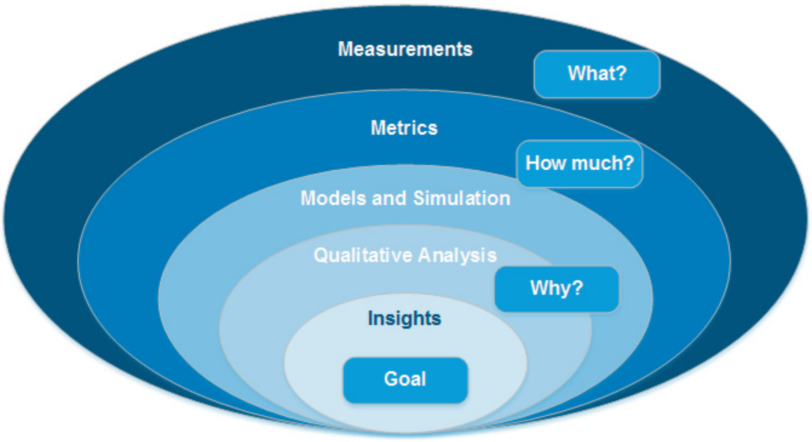


Figure 2.1: Analytics builds from measurements, and turns their data into insights with metrics, models and simulations, and qualitative analyses. Adapted from [19].

layers that go on top of each other. As pointed out by Buse and Zimmermann [21], the job of an analyst is to combine both qualitative and quantitative data, i.e., results from many levels of analytics, in forming the most complete insights.

Analytics is often defined further with a prefix noting the intent of the analytical activity (descriptive, prescriptive etc.), the specific topic of interest (health analytics, learning analytics etc.) or the object of the analysis (Facebook analytics, Twitter analytics etc.) [22]. For example, Banerjee et al. [23] and the Gartner Analytic Ascendancy Model [24] distinguish descriptive, diagnostic, predictive, and prescriptive analytics. Diagnostic and descriptive analytics seem similar considering the data they use. The difference is in the questions they answer – according to the definitions by Banerjee et al. [23], descriptive analytics answer *what happened* whereas diagnostic analytics answer *why it happened*. A three category taxonomy is commonly used as well and for example Delen et al. [25] leave the diagnostic analytics out. By their definitions, descriptive analytics include simple periodic reporting, predictive analytics assist in discovering explanatory and predictive patterns from data, and prescriptive analytics are used to determine alternative course-of-actions [25]. This three category distinction separates the resulting knowledge by time into hindsight, insight, and foresight respectively [24]. Furthermore, the Gartner model categorizes the different analytics by situating them not only in time, but also in value and difficulty dimensions as illustrated in Figure 2.2.

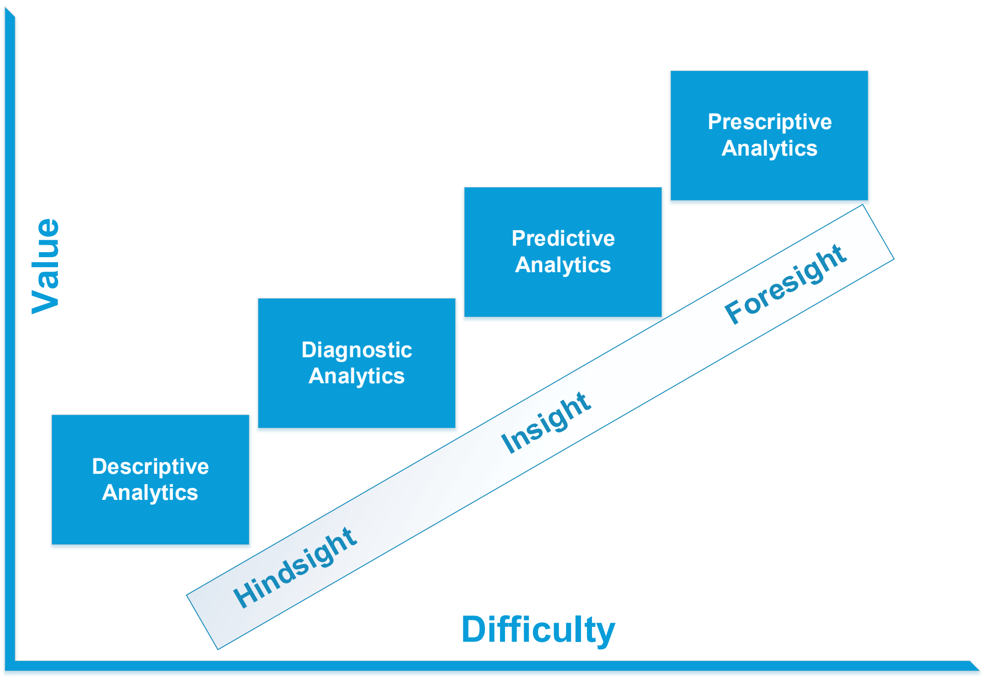


Figure 2.2: Gartner Analytic Ascendancy Model categorizes analytics by the intent of the analytical activities and positions them in time, value, and difficulty dimensions. Adapted from [24].

As a concept, software analytics scopes analytics by the specific topic of interest. Some of the business related analytics definitions, such as the one by Holsapple et al. [26], emphasize the decision-making objective of analytics. However, in this thesis we include all kinds of uses for analytics from the whole software development process. Therefore, we use the definition by Zhang et al. [27], which neatly wraps into the concept both the users and the software development process that are important aspects in the thesis:

> *"Software analytics is to utilize data-driven approaches to enable software practitioners to perform data exploration and analysis in order to obtain insightful and actionable information for completing various tasks around software systems, software users, and software development process."* [27]

Software engineering seems to be a well suited area for analytics, for example for its data rich nature [21]. Moreover, data related to software ranges from project management data to software testing reports and from business related indicators to in-the-field or run-time operation data. Measurement makes an ideal mechanism for feedback and evaluation in any engineering discipline, and in software engineering it assists all stakeholders such as developers, managers, customers and investors in understanding and controlling their software processes and products [28]. Figure 2.3 depicts the key questions addressed by analytics and lays out examples of how they fit with some of the traditional software engineering metrics and concepts.

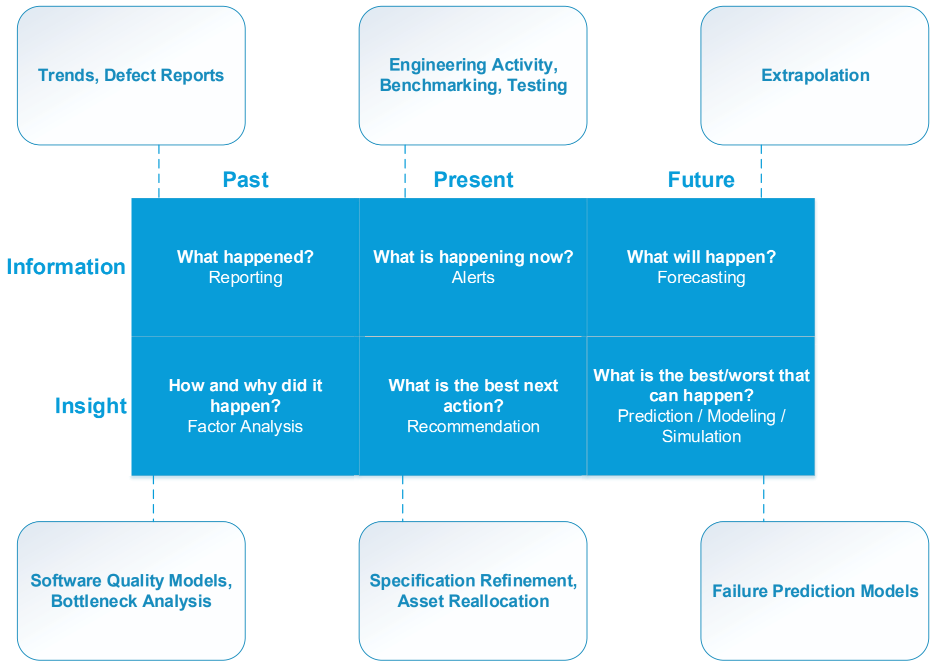

Trends, Defect Reports
Engineering Activity, Benchmarking, Testing
Extrapolation
Past
Present
Future
Information
What happened? Reporting
What is happening now? Alerts
What will happen? Forecasting
Insight
How and why did it happen? Factor Analysis
What is the best next action? Recommendation
What is the best/worst that can happen? Prediction / Modeling / Simulation
Software Quality Models, Bottleneck Analysis
Specification Refinement, Asset Reallocation
Failure Prediction Models

Figure 2.3: Time and data dimensioning of the key questions addressed by analytics. Examples of traditional software engineering metrics and concepts in outer boxes scope the more general analytics questions to the domain of software analytics. Adapted from [29] and [21].

Overall in software development, the objects of measurements have been classified to three entities: processes, products, and resources [30]. Process metrics can be related for example to the duration of the process or its activities, resource metrics consider teams, hardware etc. and product metrics are about specifications, code, or test data amongst other things. The entities have remained the same, but Kupiainen et al. [31] claim that the characteristics of measurements have changed from traditional software development's *"controlling, outsider viewpoint, tracking deliverables, setting measurable goals, following a plan, large programs"* to Agile's *"team in control, customer focus, simplicity, following a trend, fast feedback, responding to change"*. More recently, metrics such as the cycle time, indicating how long it takes from deciding that a change is needed to having it ready in production, have been considered the most critical ones in Continuous Software Development [32]. To find suitable metrics for a specific situation, well-known approaches, such as the Goal/Question/Metric paradigm (GQM) [28], can help. With GQM, the metrics are defined to answer specific questions that are set according to the goals one has for software development.

2.2 Post-Deployment Data and Knowledge Needs in Software Development

Similar to the wide variety in metrics, the set of different data types in software development is large. Therefore in the following, we will focus simply on PDD. Such data sets have been coined with different terms through the years, however. First in 1981, Plattner and Nievergelt [33] studied *run-time software monitoring*, indicating that the data sets were produced while software systems are running. This term does not leave out the data sets that are created in test or staging environments, unlike the more recent terms and definitions that emphasize more the importance of the production environment and real end-users. In 2010, van der Schuur et al. [34] defined *Software Operation Knowledge* (SOK), which they described to consist of in-the-field knowledge. In their classification, SOK was divided into knowledge about performance, quality, usage, and end-user feedback. More recently, the PDD term by Olsson and Bosch [35] was defined in 2014 to include data generated by a product after its commercial deployment. For this thesis research, we have narrowed even from PDD down to U-I data. Our definition for U-I data is similar to SOK's usage data, but as a term it emphasizes how the data are produced by users' interactions in the user-interface level. However, U-I data is not to be confused with *interaction data*, which is a term coined by Maalej et al. [36]. By their definition, interaction data covers the interactions of developers with software artifacts such as source code, documentation, command executions etc. On the contrary, we have focused in this thesis research on U-I data that considers the interactions of users, not developers, and with the system, not with any software artifact.

Web applications have been a favorable ground for using U-I data. The first approaches have focused especially on testing, e.g. [37–39]. Already in 2003, Elbaum et al. [40] studied the use of *user session data* in improving web application testing. They note that testing suites creation with data about how users operate web applications can be assisted particularly in situations where the applications evolve and have different usage profiles. Brooks and Memon [5] have found similar results in their empirical study on four open source applications. Similar to Elbaum et al. [40], Brooks and Memon [5] have formed usage profiles, *"sequences of events that end-users execute on a GUI"*, that are then used for developing probabilistic models. An algorithm then creates test cases based on the most executed user events. They found that the test suites created with their model are both smaller in size and greater in the number of faults detected when compared to test suites created straight from usage profiles.

Barik et al. [41] have described similarly how the concept of *log data* has expanded throughout the years. What was first barely an output from tracing statements assisting in troubleshooting and debugging, now has broadened to *telemetry* that continuously streams from the production environment events to the monitoring dashboards of software engineers. Nevertheless, there has been a considerable interest in clickstream data already from the early times of the Internet's commercial blooming, e.g. [42, 43]. However, the uses have been focused mainly on marketing and advertising and not as much on software engineering areas. Clickstream data has been defined in 2004 by Montgomery et al. [2] as *"information about the sequence of pages or the path viewed by users as they navigate a website"*. This definition reflects also the evolutionary status of how websites were more of a set of links whereas nowadays there is an increasing number of applications in the Internet.

At the same time, we are now seeing more examples of how organizations are measuring particularly software applications and their more complex objects, such as their features' usage [44] or even value [45–47]. In [45], the authors have come up with a feature value equation that is able to take into account different factors and their varying weights for different features. The factors, too, can be of different types. For example, a categorization into *functional, economic, emotional, and symbolic* value factors has been distinguished [48]. Tyrväinen et al. [44] have extended the scope by complementing the more traditional project metrics with data from the use of software applications. To do this, they have formed two new metrics - *"Development done to first use"* and *"Development done to value capture"*.

However, Rodriguez et al. [13] point out in their systematic mapping study that there is a lack of approaches for how to involve customers in continuous software development. In their study on Continuous Deployment (CD) [13], they have divided customer involvement in CD into five tasks: determining from whom feedback is collected, what issues feedback concerns, how feedback is collected and in which format, how feedback is processed, and how feedback is taken into account in software processes. Of these, concrete approaches for especially the last two are scarce. The need for such approaches is clear however. For example in [16], abstract steps have been laid out for advancing the use of data. The model, illustrated in Figure 2.4, is based on the similar "Stairway to Heaven" model [49], which describes how organizations advance from using traditional software methods towards continuous software engineering. In both of the models, the use of data from post-deployment time is desirable when organizations move towards the more advanced phases.

Although developers' knowledge needs in general have been studied in plenty, e.g. [50–52], only few discuss specifically the needs for user and use related data. However, we see that there are many knowledge needs in software engineering that can be supported with PDD, and these exceptions are encouraging. For example, Begel & Zimmermann [53] surveyed software practitioners about what questions they wanted answered by data scientists. They then asked the practitioners to rank the questions based on how essential and worthwhile they were. Of the 145 questions, the two top ranked questions concerned particularly the use of software: *"How do users typically use my application?"* and *"What parts of a software product are most used and/or loved by customers?"* [53]. Similarly, Buse & Zimmermann [54] identified *"Understanding Customers"* as one of seven decision scenarios in software engineering. They elaborate it as follows:

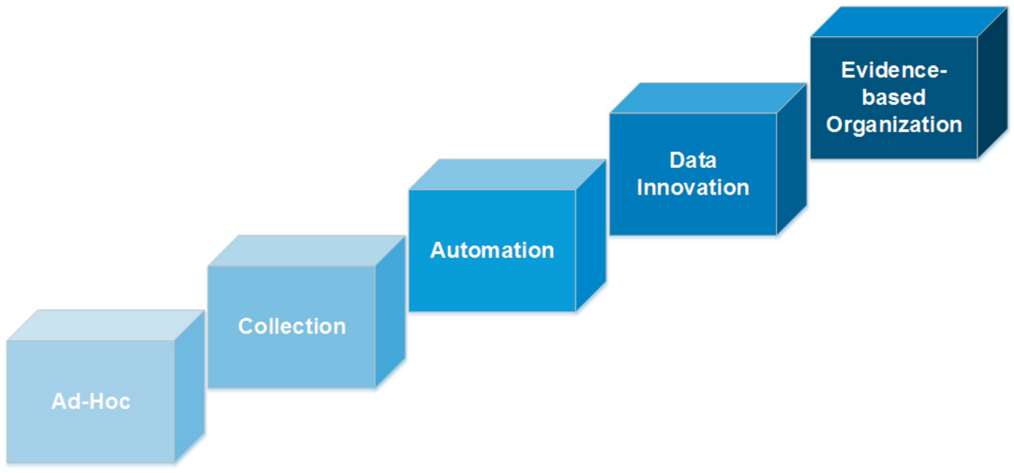


Figure 2.4: The use of data advances step by step in organizations. Adapted from [16].

> *"Analytics help us understand how a user is using our product. Are they performing tasks we expect? Performing tasks we didn't anticipate? We can determine effectiveness of features, as well."* [54]

Buse & Zimmermann [54] point out however, that the information needs range on a wide scale and that metrics with different levels of detail are differently actionable for managers and developers. In that sense, the above example of decision scenario for understanding customers could be suitable for managers, but developers might need a more detailed version. Backlund et al. [55] describe examples of such knowledge needs in a bit more detailed manner. They examined software practitioners particularly on their needs about how customers interact with an application in a single case study. They categorized the needs into four as described in Table 2.1.

Table 2.1: Examples of high detail knowledge needs about user interaction. [55]

Category	Description
Misunderstandings	Where do the end-users have issues in using the application?
Statistics	Descriptive statistics on for example what are the least/most used parts of the product
Time	When and how the end-users use the application?
Frequency	How often is the software used?

2.3 Approaches for Data-Driven Software Development

As pointed out in [6], there is always a plethora of ideas how to improve a software product. To cut down to a manageable number, the development of ideas needs to be prioritized somehow. This can be based on the opinions of people - and especially of the leaders higher in the organizational charts, i.e., the Highest Paid Person's Opinion (HiPPO) [56]. This is risky though, if the assumptions are somehow erroneous and if they can be validated only rarely. One could ask the users for what they want, but that approach has its limitations as well. As seen for example in Facebook, there can be a difference between what people say they do and what they do [57]. However, products using the Software-as-a-Service model, and connected devices in general, are now capable in increasing numbers to move from this opinion-based decision making towards data-driven software development [6]. We have distinguished three topics to support this trend, and we will cover each in the following subsections. Firstly, we will describe methods for collecting user related data. Secondly, we will take a look at methods for analyzing software data and thirdly examine the concept of experimentation systems in software development. Finally, we will present what kind of challenges are related to using data-driven approaches in software development.

2.3.1 Collecting Data from Users

For years, the human-computer interaction community has had well-established methods for collecting user related software data. For example, Holzinger [58] described thinking aloud, field observation, and questionnaires as the most common basic usability testing methods. Fabijan et al. [59] continue the similar listing by mentioning customer interviews, customer questionnaires and customer surveys. In their systematic literature review, they found that most often customer feedback is collected from direct interactions with the customers. Controversially in a multiple case study with five Finnish software companies, Sauvola et al. [60] stated how only one company had the opportunity to collaborate directly with their end-users. Many things can factor in the lack of such opportunities, e.g. business models (B2B vs. B2C), but it seems that there is a demand for collecting data from users also without a direct contact with them. To meet the demand, for example the mentioned field observation method can be conducted also as electronic observation, which Holzinger [58] refers to as *"data logging"*.

The descriptions of the feedback collecting methods by Fabijan et al. [59] are interesting in the scope of this dissertation because they have examined the methods also in terms of the collection time related to the application's phase of development. For post-development techniques and methods they list incident reports, customer pairing and bootcamps, walk-throughs, A/B testing, and social networks [59]. They have also reviewed the techniques considering the related limitations. Incident reports are available only after an incident, customer pairings require the physical presence of participants and walk-throughs can be time-consuming. Of the quantitative data collection techniques, they report A/B testing as potentially confusing for customers because of the exposure to different versions. Then again, social networks are seen to produce large quantities of data for analysis and the number of sources can be a challenge [59].

Moving on from the general approaches of user feedback collecting, there has been a number of studies focusing on single techniques for the automated collecting of quantitative data. Roehm et al. [61] first motivate their work by pointing out that it is not just the problem of HiPPO, but also the assumptions of developers about the behavior of users are actually tested and corrected only rarely . They then propose a technique of monitoring user actions by instrumentation, detecting use cases with machine learning, and finally comparing use case steps with the monitored user actions. The target is at identifying improvement points from the software system under development. They describe instrumentation as the collection technique by mentioning how it can be implemented by *"framework hooks,* *log file monitors, special monitoring code, or byte code instrumentation"* [61]. However, they do not go on evaluating these approaches apart from noting that the techniques have a varying degree of application independence and reuse possibilities.

Video games have offered one of the first places to really take advantage of user data collecting. For example, already in 2008 Kim et al. [62] have elaborated in high detail how their instrumentation system has been successfully used for improving two Microsoft Games Studios games, Halo2 and Shadowrun. However, their description of the system does not cover any technical details of how the instrumentation was implemented. Based on these two case studies, they conclude that the use of instrumentation allowed them to collect data over extended periods of time, to collect very precise data, and to do quick iterations of the game parameters [62]. Overall, it seems that the gaming industry is a fruitful ground for the use of instrumentation.

We have excluded the further reviewing of the most simplistic user data collecting techniques from this list of related work. These *data loggers* have been developed and studied in numbers, e.g.[63–65]. However, the data these techniques collect form of low level details such as mouse movements and keystrokes but lack the semantics and context of user actions [66]. On the other hand, the approach described in [66] is able to collect both low and high level of detail data. The same technique is used in [67] and it utilizes the Microsoft Active Accessibility API through the Managed Windows API[1] wrapper. At the same time, this limits the technique to use only with Windows applications.

Similar to our scope, Magalhaes et al. [68] have focused on the application level user-interactions. As they define in [68], application-level monitoring collects data about the functionality of the applications rather than just whether the applications are available or not. In addition, they have identified it from monitoring in system-level, in container-level, from end-to-end monitoring, and from log-analysis. Again similar to our work, they have used an aspect-oriented approach for developing the monitoring mechanisms. However, their work focuses particularly on detecting anomalies in the web domain and especially from performance related data. Similarly, Musson et al. [69] have focused on performance data, but their data collection technique is interesting in that they describe it as being close to aspect-oriented [70] and using event-driven API for monitoring system-level operations. However, they do not describe the technique in more detail or regard that as a research question.

All in all, there are plenty of studies related to the collecting of data automatically from users. However, the studies consider the collecting techniques only as means and not as research topics. Few have categorized such techniques and those who have, do it more as a side note. For example, Marciuska et al. [47] describe how there are two approaches for monitoring feature use. Firstly, an application can be extended to include monitoring code. Secondly, an application can be set to be intercepted by another monitoring application. Although such distinction into two is a good start, it also uncovers a research gap for the automated collecting techniques of U-I data. To the best of our knowledge, no research has been conducted that would identify different techniques in high detail and evaluate them.

2.3.2 Analyzing User-Interaction Data

Unlike the collecting of user data and its techniques presented above, the analysis methods have not been in the focus of this thesis work. However, such methods have

[1] mwinapi.sourceforge.net

been used during the work and therefore we now present an overlook on the related research fields. In its simplest form, the collected data can be analyzed with some basic statistics. For example, Groen and Koch [71] state that descriptive statistics, correlations, and visualizations are used for analyzing behavioral patterns from data. Textbooks on software metrics, such as [30, 72], give examples of these useful but general measures: Ratio, proportion, percentage, rate, six sigma, mean, median, and mode.

On the other end of the spectrum, researchers have developed also more complex methods that are geared specifically towards U-I data analysis. Focusing on a particular tasks, for example El-Ramly et al. [73] have analyzed system-user interaction traces to recover software requirements. Furthermore, the same group developed a process for discovering interaction patterns to re-engineer user-interfaces and for personalization [74]. For a more general approach, Pachidi et al. [75] developed *Usage Mining Method* of which an overview is illustrated in Figure 2.5. The method is focused on depicting how software operation data is turned into knowledge through different steps of data processing. In this sense, the method lacks the concrete intersections of how to use the gained knowledge in software processes. However, Pachidi et al. [75] offer detailed technique suggestions for the three parallel analysis methods included in Usage Mining Method. Particularly welcome for the interested practitioners might be that they have selected the techniques based on their implementability in R. For *Classification Analysis*, which they use for improving conversion rates, *Logistic Regression Models* [76], *Classification Tree Models* [77], and *Multilayer Perception Models* [78] are suggested. Similarly for *Users Profiling* they propose *Cluster Analysis* [79] and *Kohonen Maps* [80] to increase marketing intelligence and to retain old customers. *Clickstream Analysis* can assist in extracting different usage scenarios, analyzing usability and predicting the actions of users and to conduct it, the authors offer *Sequential Pattern Mining* [81], *Probabilistic Expert Systems* [82], and *Markov Chains* [83].

Essentially, the Usage Mining Method belongs to the research field of Mining Software Repositories (MSR). As Hassan [4] defines, MSR field produces actionable information about software systems and projects by analyzing data from software repositories. The repositories can be anything software related, such as historical (source code, bug, or archived communications) or run-time (deployment, execution, or usage logs) repositories. Based on their comprehensive literature survey, Kagdi et al. [84] describe MSR research with four dimensions: the software repository type (what), the purpose (why), the adopted/invented methodology used (how), and the evaluation method (quality). By these definitions, our studies could be looked as MSR research on run-time repositories (*what*) and on *why* practitioners want to use the resulting information. However, much of the research in MSR focuses on the mining methods (how), which again have only been in an enabling role in our research efforts. Although related work seems scarce for run-time repository mining focusing on U-I data, there are some studies to be mentioned. For example, Baysal et al. [85] have explored how web server logs of web browser usage can be unified with product release history. Similarly, Mattila et al. [86] introduce an approach to combine data from issue management, development and usage in single visualizations.

Moreover, the field of software visualizations offers approaches for the analysis part as well. *Software visualization* as a concept refers to the visualization of software and its development process - or more specifically of the structure, behavior, and the evolution of software [87]. To start with, one can use general visualizations, such as Gantt charts [88] or burndown charts [89], in the context of software development to assist in common topics such as project planning. Specific types of visualizations that are tailored for

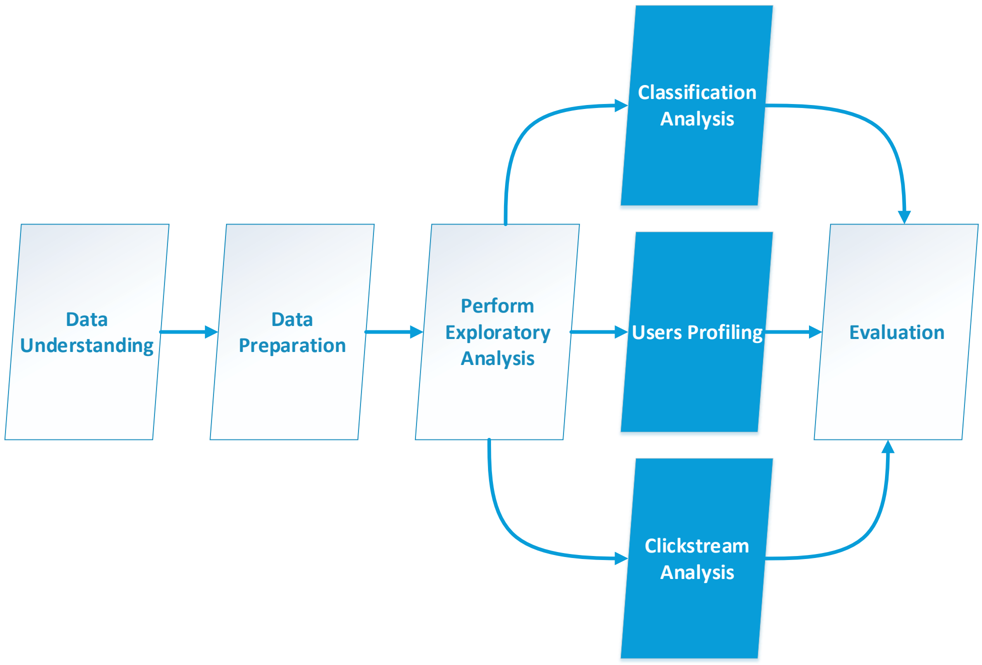


Figure 2.5: Overview of the Usage Mining Method. Adapted from [75].

software development exist as well, of course. They can be used for example for debugging [90], analyzing performance bottlenecks [91], or program comprehension [92]. However, similar to MSR methods, the specific software visualizations have not been in the scope of our research. We have created and used visualizations in the studies of this thesis, but for us the most simplest and general visualization types, such as bar charts and pie charts, have been sufficient. Therefore, we consider software visualization as an enabling and related topic for our work and have presented its basics as such.

2.3.3 Experimentation System

Over the last few years, an experiment-driven way to develop software has gained more and more attention. The approach aims at guiding subsequent development activities with factual feedback from previous software versions [3]. Already, there are some descriptions of results on the benefits of experimentation in software development: Fabijan et al. [93] list such benefits in team, product, and portfolio levels. For teams, for example, controlled experimentation can be beneficial in activity planning and in defining performance goals. They also point out that the product level benefit of incrementally improving the products is already a well discussed benefit of A/B testing. This, however, mostly considers the web domain where the technological environment has enabled the experimentation methods to become more common. For example, Kohavi et al. [94] have presented a valuable guide for controlled experiments on the web. They provide practical descriptions on critical topics such as statistical power, sample size, and techniques for variance reduction. Being a technical guideline, their approach lacks the connection to the process level activities of software development.

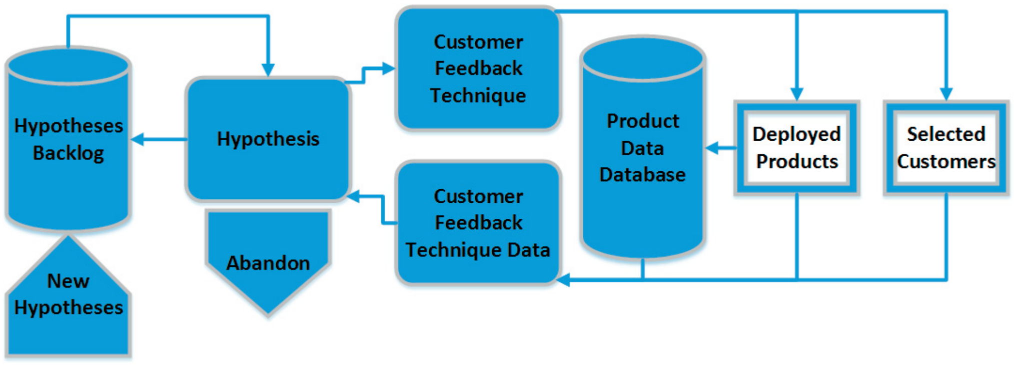


Figure 2.6: In Qualitative/Quantitative Customer-Driven Development (QCD) new software features are considered first as hypotheses that are either developed further or abandoned after experimenting them in the field. Adapted from [95].

On process level, a few methods have been introduced to incorporate hypothesis making and validation into software development. For example, Fagerholm et al. [96] described a general infrastructure, the RIGHT framework, for running continous experiments. Furthermore, Holmström-Olsson and Bosch developed the Qualitative/Quantitative Customer-Driven Development (QCD) method [95], illustrated in Figure 2.6, and the HYPEX model [97]. Similar to the HYPEX model, QCD looks at the development of new features as the validation of hypotheses and adds customer feedback techniques as methods to do that. At the same time, it unifies the process-level use of different customer feedback techniques, both qualitative and quantitative. A suitable feedback technique can be selected singly to validate a specific hypothesis, but also multiple techniques can be used for the same hypothesis validation. The method's way to combine operation data from deployed products and qualitative customer data, such as interview data, can give practitioners a valuable idea how to start using product data from the field. This is similar to Q-Rapids framework [98], which lays out a method for combining data both from development and deployment time to form software requirements. However, the methods consist of quite high level concepts and more concrete details are needed for implementation.

In contrast, van der Schuur et al. [34] have developed the SOK framework, Figure 2.7, that looks at the whole software development process including its various activities and phases. Considering the research of this thesis, the SOK framework is in that sense one of the most closely related work we found from the literature. It divides the use of SOK into five phases: *identification, acquisition, integration, presentation*, and *utilization*. The utilization phase of the development perspective is similar to our research viewpoint, and they have distinguished four activities for SOK in it. These are *informed development, usability improvements, software maintenance,* and *release management*. In addition to the development perspective, they have also looked at company and customer perspectives which can be very helpful in generating new knowledge. However, in practice the different perspectives might be becoming more and more intertwined as development teams turn increasingly cross-functional. Altogether, the SOK framework provides a detailed look into how and where software vendors can use SOK. SOK consists of more types of data than what is related to U-I data as defined in this thesis however. Considering the

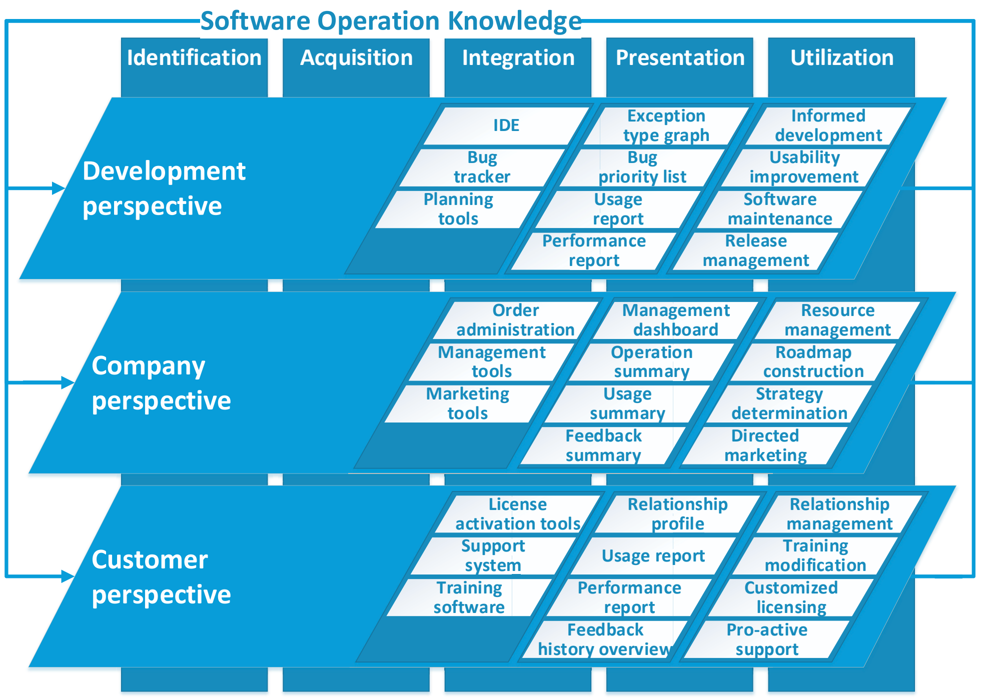


Figure 2.7: Selected details from the Software Operation Knowledge (SOK) framework, adapted from [34]. SOK framework gives examples of how and where to use SOK in software development organizations.

acquisition for instance, the techniques can vary a lot between collecting performance and usage data.

To summarize, there are already some scientific methods, models and guidelines for using PDD in software development, but few organizations are yet systematic in it in practice. This problem is noted by many authors. Van der Schuur et al. [34] found that their case study's three software vendors lack a clear plan for utilizing SOK. While they had identified the types of SOK that are valuable to them, they had only used specific tools to acquire SOK and they had no established tools nor processes to integrate and utilize SOK. Similarly, Lindgren et al. [3] describe that the state of practice is not mature, although there is a clear interest among practitioners towards the experiment-driven approach. Especially in the organizational level this immaturity is no wonder, because the change required to follow it is pervasive. For example Fabijan et al. [99] have described the evolution of continuous experimentation in software product development suitably as *an evolution over a period of years rather than a jump.*

2.4 Challenges in Using User-Related Data-Driven Approaches

As pointed out by many, e.g. [3, 60, 100], experimentation is often not systematic and continuous in practice. Since moving towards it is still tempting to many [34], there is already a body of literature about the challenges of using user-related data and data-driven approaches in software development. In the following, we will describe these challenges to draw a picture of what hinders practitioners in this movement.

Table 2.2: Challenges in adopting continuous experimentation faced by embedded systems companies [101].

Organizational	Business	Technical
HiPPO	Long release cycles	Expensive testing scenarios
Managing multiple stakeholders	Lack of sharing data in business-to-business	Real-time and safety constraints
Experts doing repetitive tuning	Metrics validation	Lack of Over-the-air updates
	Lack of data insights	Lack of experimentation tools
	Privacy assurance	

Considering the practices of experimentation in software development, researchers have categorized challenges for example into three: organizational, business, and technical challenges [101]. The more detailed challenge categories found by Mattos et al. [101] are listed in Table 2.2. Lindgren and Münch [3] describe similarly the changing of the organizational culture as a challenge in moving towards experiment-driven software development. Additionally, they found obstacles in accelerating the development cycle speed, and in finding the right measures for customer value and product success [3]. Similarly, the challenge of defining an overall evaluation metric is pointed out by many, e.g. [94, 99, 101]. In [102] Holmström-Olsson et al. have focused on the challenges of A/B testing. Also based on their study the metric alignment has turned out a challenge as they distinguished three obstacle areas: the scalability of the experiments' impacts, the alignment of the business KPIs and team level metrics, and the uncertainty of the applicability of the available solutions across domains. The challenges of the different contexts in experimentation is raised also by Bosch and Eklund [103]. They studied continuous experimentation within the context of embedded systems and pointed out challenges such as safety critical systems and hardware limitations.

With a slightly wider look into the related development approaches, we can see more challenges that can affect also the adoption of user data practices. Rodriguez et al. [13] report on the high level challenges of CD: transforming towards CD, customer unwillingness, increased quality assurance effort, and context challenges of the embedded domain. Furthermore, Claps et al. [104] and Leppänen et al. [12] describe the technical and social challenges of introducing continuous deployment with some more details. They mention how customers might prefer non-frequent releases, how there might be domain constraints, and how development organizations might be stuck with manual testing. Also Johanssen et al. [105] note the challenges of frequent releases. However, their viewpoint is that usage patterns develop only over time, and frequently changing systems produce usage data that are difficult to synchronize between the releases and the intended feedback.

The research field of requirements engineering has also approached the topic of using user-related data for their purposes. In [106], Johann & Maalej have studied the possibility of users' mass participation in requirements engineering. Their study is focused on utilizing qualitative user feedback, such as application store reviews. However, they discussed six challenges of their approach in detail, and these are interesting also in the scope of this thesis since they provide a good reflection point between the different types of feedback, i.e., qualitative vs. quantitative. For their approach, the *scalability* was a challenge because of the unstructured nature, large amount, and different quality of data. The large amount of people means also a large amount of *conflicts* between crossing opinions and demands. Still, the users have to be *motivated* to participate in the first place. That again can lead to the challenge of *representativeness* and *subjectiveness* of the participating users. In addition, the possibility of data *misuse* was seen as a challenge because false users, (lack of) data security and privacy can end up creating problems. Similar to [106], Groen [71] has discussed the challenges of crowdsourcing requirements engineering and categorized them into four main challenges. They also see challenges in the mobilization of the crowd and in informing the crowd continuously about what is done with their input. Furthermore, the understanding of the diverse feedback is emphasized as well as the privacy concerns. Finally, they also point out that the significance of the data must be somehow defined. The connection between, and validation with, usage data is proposed [71].

Using only quantitative data does not solve all problems. Rather, the research in the field of big data has identified surprisingly similar categories of challenges. For example, Sivarajah et al. [107] describe three main categories: Data, process, and management challenges. Data challenges are related to the nature of the data set, e.g. volume, variety, and velocity. Process challenges consider how to capture data, how to integrate data, how to transform data, how to select the right model for analysis and how to provide the results. Finally, management challenges raise again the topics such as privacy, security, and ethical aspects [107].

3 Research Design

In this chapter we describe the design of the research we carried out for the thesis work. In Section 3.1 we take a look at the strategies we have used for guiding our research. DS and ADR are presented and their use in our studies is covered. Additionally, we describe the research process and schedule. In Section 3.2 we go deeper into how we have conducted our research and explain the used research methods. In Section 3.3 we present the organizations we conducted our research with. Finally, in Section 3.4 we take a look at the validity considerations for our research design.

3.1 Research Strategies

The main goal for our research was to produce an actionable set of tools and methods for software teams to increase their capabilities of using U-I data in software development. Equally, we aimed at making scientific contributions in the state of the art related to U-I data and its use in software development. Our goal was therefore twofold focusing both on practical and on theoretical results. For research in the fields of software engineering and information systems such dual missions are encouraged and there are research strategies that guide research with such goals [18]. Of such strategies, we have used DS and ADR for conducting the thesis research. In the following subsections, we will go through the basics of these, and describe how we have used them in this thesis work. The research activities took place from the first quarter of 2015 to the first quarter of 2018. The schedule of the research work is illustrated in Figure 3.1 sorted per publication.

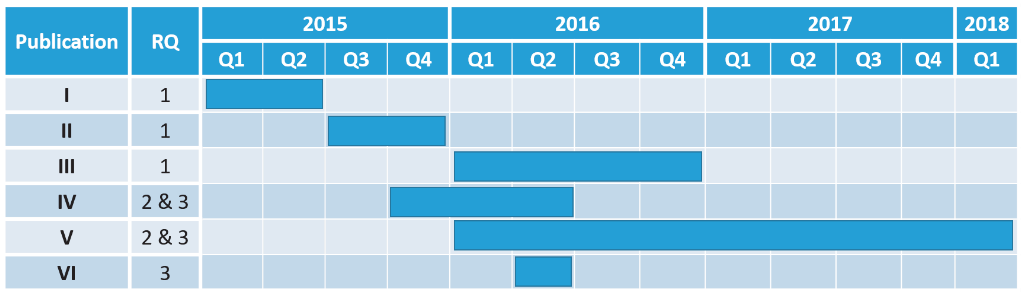


Figure 3.1: The research activities of this thesis for answering the Research Questions RQ1 to RQ3 and resulting in Publications I to VI in a quarterly timeline from 2015 to 2018.

3.1.1 Design Science

DS approaches research by building and evaluating IT artifacts. It has been adopted widely in IS research [108] and for example Wieringa [109] has presented its use also in software engineering. The aim of DS research is at the utility of the contributions although it should have implications also in the knowledge base [17]. This makes it different from both routine design and from typical research.

According to Hevner et al. [17], there are four types of artifacts that DS produces: constructs, models, methods, and instantiations. Constructs provide common language, models use constructs to represent real world situations, methods define processes, and instantiations show the implementability of the former. The purpose of the artifacts is that they should advance the organizations that use the artifacts. At the same time, codifying the results to the knowledge base makes them best practice [17].

Peffers et al. [108] present Design Science Research Methodology (DSRM) with a process model that has multiple possible entry points. The process can start with a problem-centered initiation, objective-centered solution, design & development centered initiation, or as client/context initiated. The DSRM process and its implementation during this thesis research is illustrated in Figure 3.2. We used DSRM for the first technology-oriented research segment.

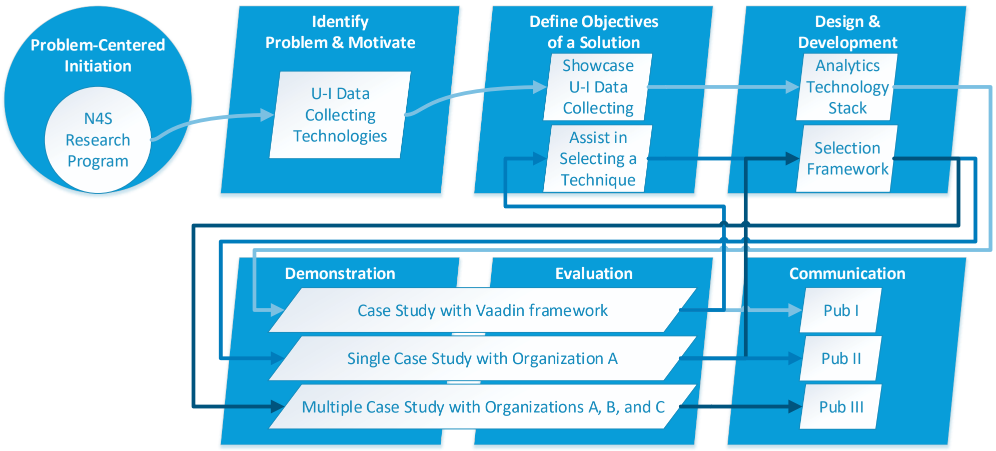

Problem-Centered Initiation
N4S Research Program
Identify Problem & Motivate
U-I Data Collecting Technologies
Define Objectives of a Solution
Showcase U-I Data Collecting
Assist in Selecting a Technique
Design & Development
Analytics Technology Stack
Selection Framework
Demonstration
Evaluation
Communication
Case Study with Vaadin framework
Single Case Study with Organization A
Multiple Case Study with Organizations A, B, and C
Pub I
Pub II
Pub III

Figure 3.2: The Design Science Research Methodology process (adapted from [108]). The entry point is illustrated with a circle and the activities with parallelograms. Our implementations are seen in white background while the concepts are presented in blue. Our three iterations are depicted with blue arrows, the first with the lightest and the third with the darkest.

In our case, the research program[1] provided us with a clear problem area to focus on. The aim of the program was to study real-time experimental business models and to provide capabilities for instant value delivery based upon deep customer insight [110]. We initiated the work for this thesis by studying the collecting of U-I data and the related technologies. This way, we went through the first iteration of DSRM process and produced

[1]http://n4s.dimecc.com/en/

the first artifact, the design for the unobtrusive analytics technology stack presented in Publication I. The demonstration and evaluation phases of the DSRM process were conducted with a case study, where we used the analytics stack with Vaadin framework[2] and its demo application[3]. This work was done in the early 2015. The objective of the first iteration was to produce a showcase artifact. With that we could demonstrate the collecting of U-I data and its utility for interested organizations in the N4S program.

After a successful demonstration with the first artifact, we got to conduct a single case study with organization A in the fall of 2015. There, we defined a new objective for the second DSRM iteration. The selecting of an appropriate U-I data collecting technique was not an easy task. Therefore, we decided to assist this by developing the second artifact, i.e., the selection framework. This was presented in Publication II. Then, we were able to conduct a multiple case study with organizations A, B, and C from February to December 2016. The objective of this third iteration remained the same as in the second iteration. However, we were able to focus on the evaluation of the artifact. This resulted in a few adjustments and refinements in the selection framework and these are presented in Publication III.

3.1.2 Action Design Research

Similar to DS, ADR seeks to generate prescriptive design knowledge by developing and evaluating IT artifacts [18]. However, ADR adds an organizational setting to the research. It mixes the evaluation of the artifacts tighter into the method, whereas DS had the evaluation as its own separate phase. Like DS, ADR addresses the class of problems rather than solving immediate case specific problems [18]. The main difference between DS and ADR is in the researcher intervention. The ones without researcher intervention in organization could be looked as DS. With such studies, the intervention can still happen in the evaluation phase. The studies with an intervention, on the other hand, appear more like ADR.

We used ADR for guiding our work in the second research segment. The study is presented in Publication V. The built and evaluated IT artifact of that study was the U-I Data Utilization Method. Although the case organizations were the same as in the previously presented DS guided multiple case study, we intervened with the work of the case teams more clearly. The teams had either no or very little previous experience with using U-I data for software development, and our study with them started such efforts. The U-I data utilization method models the work done in and with the case teams during the research period of February 2016 to February 2017.

3.2 Research Methods

To conduct the research guided by the chosen research strategies and to add to the knowledge base of software engineering research, we mainly used the case study method. In addition, we conducted one study with the survey method and one publication (Publication VI) was a concept paper without a research method. In the following, we will present these methods that we used for conducting the research of this doctoral thesis.

[2]https://github.com/vaadin/framework
[3]https://github.com/vaadin/dashboard-demo

3.2.1 Case Studies

Yin [111] describe case studies as rich and high detailed descriptions of specific instances of a phenomenon, and that they are usually based on multiple data sources. Runeson and Höst [112] distinguish between four types of case studies by their purposes. *Exploratory* studies are aimed at finding out what is happening to generate knowledge and to produce new ideas and hypothesis for future research. *Descriptive* studies focus on portraying a situation or a phenomenon. *Explanatory* studies seek explanations to a situation or problem. They are mostly but not necessarily of the form of causal relationships. Confirmatory case studies are also seen as explanatory. Finally, some case studies are intended to *improve* a certain aspect of the studied topic [112].

During the doctoral thesis research, we used many of the different types of case studies. All of the studies in the first research segment were case studies, but all of them were of different type. The first one could be defined as an improvement study since it investigated the characteristics of a new U-I data collecting technique. The second was an exploratory case where we examined a specific situation in one organization. We used its results as a basis for a new development phase for the research artifact. The third case study was then an explanatory study that was aimed at evaluating and confirming that artifact. Finally, after a year from the ADR research period that led to the development of the U-I data utilization method we studied the same case teams on how they had actually utilized U-I data on their own. This could be looked at as a descriptive case study on its own, although it was published as a part of the whole ADR study in Publication V.

3.2.2 Data Gathering and Analysis Methods

We used a wide scale of research data gathering methods during the thesis work. Time-wise, the most used method was to organize, participate in, and/or examine the memos of different workshop meetings. Three types of workshops were held during the studies either with or by each case team. Firstly, the researchers motivated the teams to consider U-I data collecting as a new possibility to them and discussed the options for collecting techniques. Secondly, two of the teams had internal brainstorming workshops for coming up with objectives for U-I data. One team had such a session with the first author of this thesis. Thirdly, after collecting the U-I data in each case, either a presentation of its results or a workshop for analyzing the results was held together with the team members and the first author. All of the above workshops were of unstructured nature. In addition to the research data gathered in these workshops, the first author of this thesis had designated work desks in the same rooms where the software teams were working in cases A and B. This allowed us to explore the contexts of the study thoroughly and to exchange information also informally with the team members. For example, this setting allowed us to give concept presentations within the organizations.

Additionally, we used a questionnaire survey for the study presented in Publication IV. The questionnaire was sent via email to four Finnish software consulting companies small to medium in size. The questionnaire included five open-ended questions about PDD and a few about the background of the respondents. 25 responses were given. We also interviewed the case teams' practitioners formally for the final part of the ADR study presented in Publication V. To allow a broad range of answers, the interviews were semi-structured and we mainly used open-ended questions. Altogether we had three interviews, one for each case. Two researchers were present in all of them and the number of interviewees was three in case A and one in cases B and C. The interviews were recorded and then transcribed by a third party company with a basic transcription level where e.g. filler and repetitive words were left out.

For analyzing the results, we used thematic coding in both the interview study, the questionnaire survey, and in analyzing the workshop memos. However, there are different approaches for thematic coding as described by Hsieh and Shannon [113] and we used two of them. For analyzing the final interviews from January 2018, we used conventional content analysis [113] to form categories of the challenges the teams faced with U-I data utilization. This method allowed the categories to flow straight from the data, which was important for recognizing also new kinds of challenges. In this case, the analysis was done by three researchers. Similar method was used for the questionnaire survey results, although its results were analyzed by two researchers.

The results from the questionnaire study were used as a basis for the analysis conducted for validating and refining the categorization of U-I data analysis and use objectives. For this reason, we used the directed approach to qualitative content analysis. The goal of such analyses is to validate or extend conceptually a theoretical framework or theory as described in [113]. The U-I data objectives from the brainstorming sessions were extracted to a spreadsheet, and we also marked the information of which case it came from and if it was selected for the collecting or not. Then, one of the researchers labeled each objective according to the PDD analysis and use objective categorization created in the questionnaire survey, presented in Publication IV. For triangulation purposes, ten quotes were then given for similar labeling to two other researchers.

3.3 Participated Software Teams

To study classes of problems rather than solving barely immediate organizational problems, we wanted to do research in more than one authentic setting. Therefore, we selected case organizations of different sizes and teams with their software on different technical environments. On the other hand, the focus of the research on software teams that were not yet accustomed to the use of U-I data limited the number of the potential cases.

Furthermore, readiness to try to collect and use such data was required from the case organizations. The selected cases needed to be accessible to us as researchers and be open enough to make publishing the results possible in a reliable manner. In addition, the number of the cases selected for the research was affected by the fact that we had to spend considerable effort in each case. These limited the number of selectable cases to few, and finally three software teams from three organizations were selected to participate.

Organization A is a large international telecommunications company. The software team that was involved in this thesis research consisted of eight members. However, the boundaries between the teams are quite flexible as employees work for many products. The team members had titles of software architect, UX designer, software developer, and line manager. Their products consist primarily of software in the field of network management, and these range from Java software to web based systems. New versions of their products are released usually a few times a year.

Organization B is a medium sized software company in Finland. At the start of the research for the thesis, it had around 300 employees and offices in three major cities of Finland. The company primarily develops software in projects for their customers as ordered. The software team involved in our studies, however, develops their own SaaS solution. During the study periods, the team was spun-off to a company of its own. In contrast to case A, company B releases new versions of their product to the end-users far more often − usually biweekly. Their software team consists of seven members with titles such as product owner, UX specialist, software architect, and software developer.

Organization C is a research and education center of circa 10000 students and 2000 employees. The case C software team is part of a research group who have specialized in embedded systems design. They have developed a software solution for ASIC, FPGA and embedded systems design. The software has created traction from users worldwide, mainly from the USA and from Central Europe. The development team has four members with the titles software developer, software architect and business architect. The tool is an installable software system released three to four times a year.

Organization A participated in our studies published in Publications II, III, and V. Organizations B and C participated in studies published in Publications III and V. In addition, Publication IV was based on a study, where we surveyed four Finnish mid-sized software consulting companies.

3.4 Research Validity

We discuss the validity of our research and its design from the four perspectives proposed by Runeson and Höst [112].

Construct validity reflects how the studied measures actually represent their real-life counterparts. For example, the topics discussed in an interview should be interpreted the same way by the interviewer and the interviewee. The interview study of this thesis was semi-structured and the questions were open-ended to mitigate such a threat. This gave the interviewees and the interviewers a chance to define topics and questions whenever there was anything unclear. Similarly, the questions in the survey study were open-ended, which made it possible to the respondents to elaborate their answers. However, in such a study setting the respondents could not get our feedback if they had something to ask. To mitigate the threat of misinterpreting the questions in the survey, one researcher made an initial draft of the questionnaire and then two other researchers refined it to its final form. In our case studies and in the ADR study, the study subjects were always given the chance to review the study results and correct them where necessary.

Internal validity is concerned with examining causal relations. In this research, the only study that was set to find out explanations for a phenomenon was the multiple case study for the reasons of choosing a U-I data collecting technique. However, even in that study we emphasized that our concern was to find out different types of factors rather than to measure the exact statistical significance of them. In that case, the threat to internal validity is still clear and there is the possibility that some factors were left unknown.

External validity reflects the generalization of the results and how interesting the findings are for other people than for the ones in the studied cases. Considering the effort we had to spend in each of our cases, the number of case teams had to stay quite low (3). Similarly, the responses to the questionnaire study were collected from a small number of companies (4). In this sense, the results of this thesis research are not automatically generalizable for any software developing team. However, the case organizations were highly different in their characteristics among each other except for having no or very little experience in using U-I data in software development. Given the trend of data-driven approaches and the novelty of the field, we believe that the results can give interesting insights to a wide audience.

Reliability considers how dependent the study is from the specific researchers. In the case of this research, the bias from using one researcher for much of the work is obvious. To mitigate such threats, we used a group of researchers whenever possible. In data gathering phase, the interviews were conducted by two researchers and they were transcribed by an independent professional transcription service company. The meeting notes that were used as the data source in the multiple case study were written down by both the researchers and the practitioners. Similarly in the analysis phase, several researchers were involved. In the survey study, the analysis of the U-I data objectives and the related challenges was done by two researchers. Then, the ADR study analysis of U-I data objectives was conducted with a third and a fourth researcher and the challenge analysis with the third and a fifth researcher. In addition, there are reliability concerns in the analysis methods. Hsieh and Shannon point out the use of directed approach to qualitative content analysis is biased towards finding supportive rather than non-supportive evidence [113]. We used the directed approach in our ADR study for the U-I data objectives. To mitigate the threat, we used triangulation on the data sources. The original categorizations of the U-I data objectives were based on the survey study with a questionnaire, whereas in the ADR study we used the workshop meeting memos.

4 Results

This chapter describes the results of the doctoral research. Sections 4.1 to 4.3 present the results per research question (RQ), and Section 4.4 summarizes the contributions per publication. Firstly, we drawn results from Publications I, II, and III that are related to the technical RQ1. Secondly, we present results from Publications IV & V, which consider the U-I data objective and utilization focused RQ2. Thirdly, we first present the results from Publications IV, V and VI and then form a new synthesis of the opportunities of U-I data use in software processes. Finally, we present the results from Publications IV & V where we have identified the challenges in U-I data utilization. The synthesized opportunities and the studied challenges answer the final RQ3. The linking between the research segments, research questions, publications and contributions is summarized in Table 4.1.

Table 4.1: Linking between the research segments, research questions, publications, and the main results

Segment	RQ	Publications	Results
U-I Data Collecting Techniques	RQ1	I, II & III	• Framework for Selecting U-I Data Collecting Techniques • Identification and evaluation of five collecting techniques
Using U-I Data in Software Development	RQ2	IV & V	• Categorizations of U-I data types and objectives for analysis and utilization • Utilization method for U-I data
Opportunities and Challenges of U-I Data Utilization	RQ3	IV, V & VI	• Synthesis of U-I data objectives and targets of iterative SW development methods • Categorization of the challenges in starting the use of U-I data for SW development.

4.1 U-I Data Collecting Techniques

The first research question was formed as *RQ1. How to select a technique for collecting user-interaction data?* We addressed this question with studies presented in Publications I, II and III. First, we designed an analytics stack that showcased the idea of collecting U-I data unobtrusively from Java applications. Since that stack was intended mainly for studying one specific technique, we then identified on a conceptual level four additional techniques for collecting U-I data automatically. To get to a more practical level, we then studied the challenges organizations face when starting the collecting of U-I data. Afterwards, we extracted the challenges into criteria for evaluating data collecting techniques.

To assist practitioners in getting started with U-I data collecting, we joined development teams in their daily work in three cases. For each of these three teams, we presented the five collecting techniques. Then, together with the teams, we selected a technique that we thought was the most suitable in their case. This selection process used the evaluation criteria we had formed beforehand. We described the process and included it in *the framework for selecting U-I data collecting techniques* that is presented in the following subsection.

4.1.1 Framework for Selecting U-I Data Collecting Techniques

Table 4.2: Evaluation criteria for U-I data collecting techniques

Criteria	**Description**
Timeliness	Time-wise considerations on the availability of the data the technique produces.
Targets	Technique's support for collecting multiple types of data and/or for varying purposes and stakeholders.
Scalability	Describes the work effort required from the developers in implementing additional collecting places to the source code.
Overhead	Effects of the technique to the performance in the field and possible downtime during implementation.
Sources	Flexibility of collecting data from different sources, such as various platforms, applications, or from applications of different languages.
Configurability	Considerations on how configurable the technique is, e.g. switching the collecting on and off and between different types of data.
Security	Trustworthiness of the organization and of the solution behind the technique.
Reuse	Considers if the technique provides a one-time solution or if it can be reused easily with another application.
Change	Describes the levels of change required to the architecture and/or environment of the application.

The framework for selecting U-I data collecting techniques consists of two parts. Firstly, it includes a set of criteria for evaluating the different U-I data collecting techniques. Secondly, we have designed, refined and validated a method that describes the steps of selecting a suitable U-I data collecting technique.

The first published evaluation criteria set was presented in Publication II. At that point, the set was based on a literature review and a case study with one company. To make it more actionable for practitioners, we then used the set with three software teams in the actual choosing of a U-I data collecting technique. Based on these three cases, we then refined the criteria set where necessary as described in Publication III. Consequently, we claim the following set of criteria, presented in Table 4.2, validated within the case study teams.

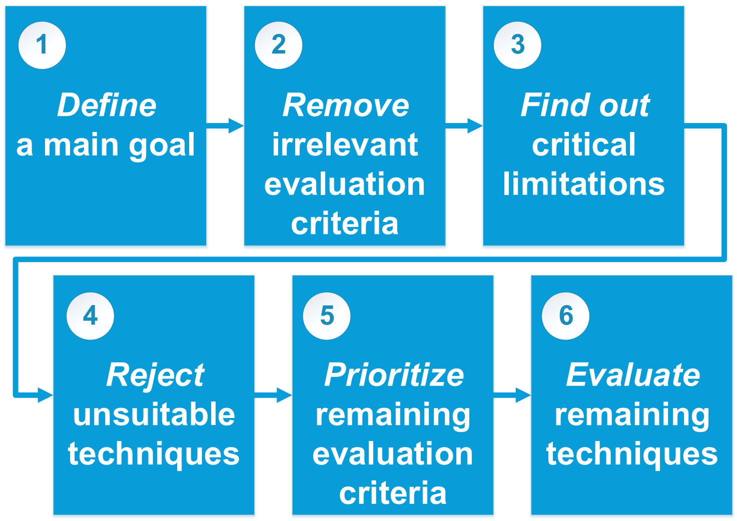

1
Define a main goal
2
Remove irrelevant evaluation criteria
3
Find out critical limitations
4
Reject unsuitable techniques
5
Prioritize remaining evaluation criteria
6
Evaluate remaining techniques

Figure 4.1: Method for selecting U-I data collecting technique. Adapted from Publication III.

The second part of the framework, i.e., the method for selecting U-I data collecting techniques, is summarized in Figure 4.1. We developed it similarly to the set of evaluation criteria in the same Publications II and III. The method consists of six steps, which describe the activities we went through with the case teams in the processes of selecting a suitable U-I data collecting technique. The steps are as follows:

1. **Define a main goal.** A clear vision of why the collecting technique needs to be implemented is a critical part of the selecting method. Defining a goal based on such vision is thus arranged as the first step in the method. It guides the decision making in the rest of the steps. For example, a goal can be defined as *"Try out A/B testing as a one-time experiment in one application"*.

2. **Remove irrelevant evaluation criteria.** Not all of the evaluation criteria are required in every case. To make the focusing on the important ones somewhat easier, the irrelevant can be removed in this step. At the same time, the step guides one to familiarize themselves with the evaluation criteria. Continuing with the example from step 1, evaluation criterion such as *"reuse"* could be removed. It would make no sense to invest in a reusable implementation of a technique if it is intended for a one-time solution.

3. **Find out critical limitations.** Before more detailed evaluations, the critical limitations to the collecting techniques by the application, environment, and organization need to be examined. These include case characteristics, such as the used programming languages, privacy regulations, and production environment technologies. For example, a regulation of not storing user related data outside a specific geographical region can set a critical limitation to the selecting of collecting technique.

4. **Reject unsuitable techniques.** At the latest at this step, one should familiarize themselves with the available U-I data collecting techniques. The step is highly based on step 3. It uses the emerged limitations in analyzing which of the techniques can be removed at this point. Continuing the example of step 3, a collecting tool developed by a third party might need to be rejected if it remains uncertain if that party always keeps the data within the suitable region.

5. **Prioritize remaining evaluation criteria.** This step is highly intertwined with step 1. The defined main goal for the U-I data collecting sets the stage for this step and assists in deciding how the remaining evaluation criteria should be prioritized. Again referring to step 1 and the example of a one-time collecting experiment, a software team might prioritize a low level of change required to their software architecture over a low scalability of the technique.

6. **Evaluate remaining techniques.** At this final point, the remaining techniques are evaluated in terms of the prioritized set of criteria. The result of taking all the steps should be an understanding of how the collecting techniques suit the case at hand.

The presented framework is not dependent on specific techniques for U-I data collecting. Consequently, it can be used also with techniques that have not been yet studied by us. However, we have identified five collecting techniques both to assist practitioners in a more concrete manner and to test the framework in a practical context. In the following subsection, we present these identified techniques with the help of the framework.

4.1.2 Identification and Evaluation of Five Collecting Techniques

We have started the identification of techniques that are suitable for collecting U-I data in our study presented in Publication I. At that point, however, we were studying only one of these techniques. In the study presented in Publication II we identified four more techniques and evaluated them on a conceptual level using the criteria of the selection framework.

During the studies of Publication III, we tried out three of the techniques also empirically with the case teams. To select the most suitable technique in each case, we evaluated all five of the techniques within each team's context. We then extracted the overall evaluations of each technique by each criterion. For each criterion, the techniques were given a plus (+), a minus (−), or a plus/minus (+/−) sign for either supporting, limiting, or not having clear support or limitations respectively. Additionally, we marked every evaluation with an exclamation point (!), if the technique was seen to have either a significant enough limitation for its rejection or support for its selection in terms of that criterion. The evaluations are presented in Table 4.3. In the following, we describe each of the techniques and their evaluations focusing on the accented points:

- **Manual Instrumentation** In the most simple case, a developer inserts additional statements to each distinct place of the source code where data is wanted to be collected from. This technique is the easiest to implement in that it does not require any *changes* to the software architecture. Nonetheless, it allows data to be collected very flexibly. Any place of all the accessible source codes can be instrumented, making the technique support the *sources* criterion. At the same time, since only

Table 4.3: Summarized technique evaluations. Adapted from Publication III

	Techniques				
Criteria	*Manual*	*Tools*	*AOP*	*UI Library*	*Exe. Env.*
Timeliness	+	+/−	+	+	+/−
Targets	+	−	+!	−	+/−
Scalability	−!	+	+!	+	+!
Overhead	+	−	+/−	−	−
Sources	+	−	−	−	−
Configurability	+	−	+	+	+/−!
Security	+	+/−!	+/−	+	+/−
Reuse	−	+	−	−	+!
Change	+!	+	−!	−!	+

+ = Supports selecting
− = Technique has limitations
+/− = No clear support nor limitations
! = A possible reason for rejecting or selecting

the desired places in the source code are instrumented, the *overhead* is limited. The technique sets no restrictions in terms of *timeliness, targets, security, or its configurations.*

On the down side, the possibilities to *reuse* the collecting points are limited. In addition, the more places are decided to be instrumented the more work effort is needed. After a while, the maintenance of all instrumented places can get heavy, making the *scalability* of the technique a possible reason for its rejection.

- **Tools for Automated Instrumentation** There are multiple tools that can automate the instrumentation of the source code. Many of the characteristics are dependent on the specific tool. Nonetheless, common to them is that they free the developers from the manual work increasing the *scalability* of the solution and its *reuse* possibilities.

  However, the automated instrumentation tools are often *targeted* at few specific things. They produce data from distinct *sources* and their *overhead* on the performance can grow rapidly if they are not easily *configurable.* This, along with the *timeliness and security* of the technique, go hand in hand with the specific tool. *Security* can be crucial especially if the tool is developed by a third party.

- **Aspect-Oriented Programming** With the help of aspect-oriented programming, the original source code can remain the same. The collecting parts of the code can be woven to the target program during compile time, leaving the original code untangled and making the technique unobtrusive. This is an advantage considering the maintainability of the applications. However, the overall *change* required in start using AOP paradigm, was seen as a significant reason for rejecting the whole technique in our studies. The availability of AOP libraries is similarly critical, reducing the number of possible *sources.* Likewise, the *reuse* options are limited to the programs that are developed in a similar enough fashion.

  On the other hand, the expressive power of AOP increases the flexibility of the technique to the level of the manual instrumentation. Since the collecting happens in the source code level, the *timeliness, configurability,* and *security* are under the control of the developer. This increases also the capability to focus on specific but various *targets*. Since the instrumentation work effort is reduced, *scalability* of the technique is high.

- **Alternative Implementation of a User-Interface Library** User-interactions happen often through standardized components of UI libraries. By alternating the implementations of the components in the libraries, data collecting elements can be included in those external parts and therefore outside the actual source code. With this technique, the *timeliness, configurability, and security* remain under the control of the technique's implementer. The *scalability* is high because no additional work is required after alternating the original library.

  On one hand, there is an increased possibility to collect extra data from unnecessary *sources* causing performance *overhead*. On the other, some data types and *targets* can be still out of the reach of these libraries. The *reuse* of the technique's solutions can seem easy considering how much reuse there is for the original libraries. However, the libraries are often developed by third party vendors. When they come up with new versions of the libraries, the data collecting needs to be implemented again. Moreover, maintaining just two different versions libraries - with or without data collecting - can add up to too much *changes* to the software architecture.

- **Monitoring the Execution Environment** With a programming language, such as JavaScript, it is possible to monitor the execution environment and implement the data collecting using that. This technique is somewhat similar with the AOP technique in that it allows the implementer to be in control of the *timeliness, targets*, and *security* of the solution. *Configurability and security* are similarly depending on the implementer of the technique. Nevertheless, since no manual instrumentations are required the *scalability* is always high.

  On the other hand, the technique can have limitations in terms of the *overhead* it produces and the *sources* it can include. In our studies, presented in Publication III, we did not discover such deficiencies. For those studies, we developed a tool that implements this technique. The tool is available in GitHub[1]. The possibility to *reuse* the tool was significant for its selecting in of the cases of the study. In addition, the technique usually does not require major *changes* to the software architecture of the target application.

4.2 Using U-I Data in Software Development

The second research question was formed as *" RQ2. How to use user-interaction data in software development?"* To answer this, we started by surveying software development teams. We conducted a questionnaire study, presented in Publication IV, in which we analyzed the software teams' views on the use of PDD in their work. We identified different types of analysis and use objectives for U-I data and categorized them. To get a more in-depth look at how practitioners wanted to use U-I data, we continued the work with a hands-on approach in three cases. This resulted in refined categories presented in Publication V.

In the same study, Publication V, we examined what kind of activities software teams have when they start the use of U-I data in their work. We modeled those activities with

[1] https://github.com/ssuonsyrja/Usage-Data-Collector

a design of U-I data utilization method. The study context was the same as with the studies of selecting the collecting techniques, presented in Publication III. We propose that the method can be used for guiding other software teams in similar tasks as well. After all, the design process of the method was cyclic and some of the experiences from previous cases can already have had a guiding effect on the later ones.

4.2.1 Categorizations of Objectives for Analysis and Use of U-I Data

During our studies, we have categorized two types of objectives for U-I data. Firstly, we identified different kinds of analyses that practitioners are aiming at. Secondly, we examined what types of uses practitioners had for U-I data. Understanding both of these types of objectives is crucial in assisting practitioners to start utilizing U-I data in their work. In the following, we go through the two categorizations of objectives.

The categories of **analysis objectives for U-I data** disclose what practitioners want to do to the data they collect. Defining the analysis objectives assists in defining what kind of data needs to be collected. The analysis objectives for U-I data are as follows:

- In **value analysis**, the objective is to estimate a value of a feature or a function based on how it is being used in the field. In the most simple case, the high rate of use indicates the high value of the feature. However, other factors from U-I data can be added to the evaluation as well, such as who is using the feature, where is it being used and at what time.
- With **pain point analysis**, the objective is to identify or validate obstacles that the users of a system might be encountering. For example, the developers might get feedback via email from a single users about a difficult user-interface function. By collecting U-I data, the developers can analyze if a larger portion of the user base is affected by the same thing.
- **Use path analysis** dissects what was the order in which features where used. The most common routes taken by the users can form a solid baseline for analyses. This can assist practitioners in realizing what the users are usually doing with the system. At times the abnormalities that have happened might be the more interesting findings.
- The objective of **user profiling** is to segment the users of the system. The segmentation can be done with various dimensions, such as the expertize of the user or the technology they are using. For example, profiling users based on what browsers they are using can help developers in estimating how valuable it is for them to widen their technical support for additional browsers.

The categories of **use objectives for U-I data** tell where and why practitioners want to use the collected data. Identifying what kind of uses there are for U-I data helps in understanding how, at what point, and who can benefit from collecting U-I data. The use objectives for U-I data are as follows:

- The objective of using U-I data for **informed feature development** is to support decision making of how or whether features should be developed. Insights from U-I data can be used for assisting the practitioners in their daily work, for example in understanding if the users are using the software as intended by the development team.

- Using U-I data for **requirements validation** is to target at making sure the specified requirements are valid for the users in the field. This objective highlights the difference between customers and users. For example, the usage of an MVP can validate the developer's hypothesis that there exists a market need for a certain solution.

- **Resourcing and prioritizing** decisions can be supported with U-I data. For example, the value of a feature based on its usage can be taken into account while prioritizing the bug fixing tasks in a sprint backlog. Similarly, resourcing decisions such as how many people should be working on the task or how much computing space is reserved for a microservice can be based on U-I data.

- In our studies, the development of **user experience** was the most common objective for using U-I data. It is similar to the informed feature development category, but focuses specifically on UX related development.

4.2.2 Utilization Method for U-I Data

The utilization method for U-I data consists of eight *activities* that are grouped into three **steps**. The activities summarize what the software teams in our cases did as they started the use of U-I data for their work. We divided the activities into the three steps based on when the activities occur, and also to form logical phases that can be recurring on their own. The method is illustrated in Figure 4.2 and the steps and their related activities are explained in the following.

The first step of the method is called **Proof-of-Concept**. Like the name entails, the motivation for the step is to produce a prototype for collecting U-I data. Since that requires using a technique for collecting, the step is intertwined with the method for selecting U-I data collecting technique. The first activity assists also in the technique selecting, however. When one takes time to first *brainstorm the analysis and use objectives*, they provide valuable information also for defining a main goal, i.e., the first step of the method for selecting U-I data collecting technique. The second activity is then to go through the rest of the process, and *select a collecting technique*. The selected technique is then used for creating a proof-of-concept solution for the U-I data collecting. The intention is to *test the collecting technique*, so that it can produce the kind of data that is needed. This testing is also the final activity of the step. Based on its results, one can either move on to the next step or take the first step again to test a different collecting technique or objectives.

The second step is to **Aim & Deploy**. The first activity of brainstorming objectives should result in an excessive number of ideas that might need to be trimmed down in this phase. This can be based on various things such as the selected collecting technique and what it can produce or which of the objectives are prioritized in a certain time-frame. Nevertheless, one needs to *select the analysis and use objectives* that the data collecting will be based on. After such selection, one has to *define the collecting points*, i.e., the parts of the source code from the target application that are affected by the collecting technique. The execution of this fifth activity depends on the selected collecting technique, and not all techniques even allow specific modifications. In a common case, however, the proof-of-concept prototype from the first step can be extended to include all the collecting points required in the objective selecting. The step should result in a collecting solution that is integrated into the target application and that can produce all the required U-I

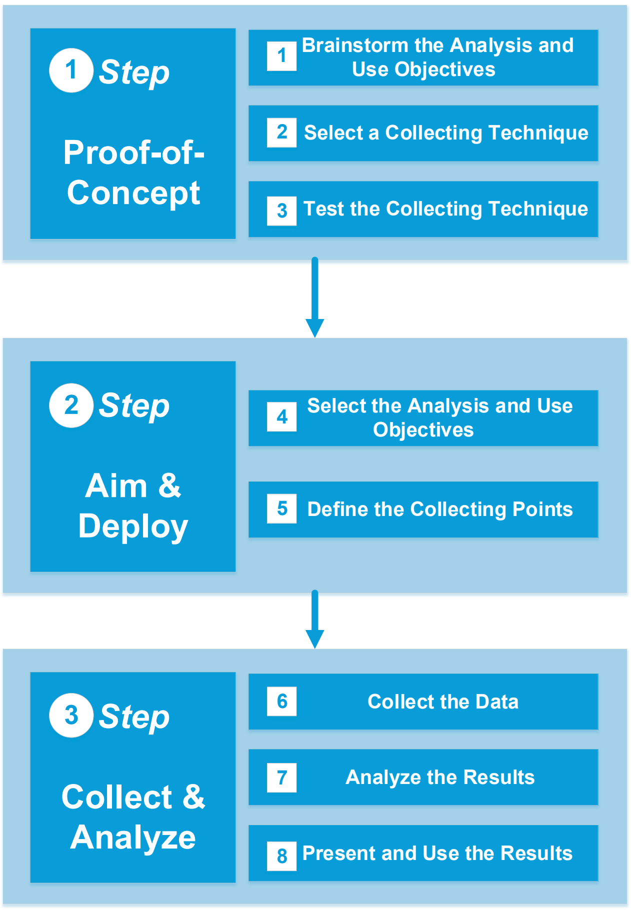

1 Step
Proof-of-Concept
1 Brainstorm the Analysis and Use Objectives
2 Select a Collecting Technique
3 Test the Collecting Technique
2 Step
Aim & Deploy
4 Select the Analysis and Use Objectives
5 Define the Collecting Points
3 Step
Collect & Analyze
6 Collect the Data
7 Analyze the Results
8 Present and Use the Results

Figure 4.2: Method for utilizing U-I data in software development. Adapted from Publication V.

data. After completing this step, one moves on to the next step rather than takes this step again, which is different for this step than the two others.

Finally, the third step of the method is to **Collect & Analyze**. The step consists of three activities: *collect the data, analyze the results, and present and use the results*. For collecting the data, a certain time-frame should be defined. Additionally, the collecting can be restricted based on a number of things such as collecting from a specific user group. These can be changed also and the whole step can be taken a number of times. Altogether, the execution of the activities in the step vary a lot depending on the previous decisions and the case context. The important phase, however, is to complete also the final activity of presenting and using the results. This way, the collected U-I data ends up in use. At the same time, its value and needs for new utilization rounds can be estimated.

4.3 Opportunities and Challenges of U-I Data Utilization

The third research question was formed as: *RQ3. What kind of opportunities and challenges are there in the utilization of user-interaction data?* We answer this research question in two parts. Firstly, we make a synthesis about the targets of software development approaches and objectives for using U-I data. Secondly, we study the challenges of utilizing U-I data. Together, these two parts assist in forming an understanding of the opportunities and challenges of integrating U-I data into software development processes.

For the opportunity synthesis, we started by studying the targets of different software development approaches and presented them in Publication VI. Now, in the following subsection, we use those results and combine them with the categorized objectives presented in Publication V and in Subsection 4.2.1. We aim to identify how U-I data could be used to support the targets of the iterative development cycles.

For the studies on the challenges, we first surveyed the opinions and experiences of practitioners. We divided the challenges into three categories and subsequently each into three subcategories. The related results are presented in Publication IV. We continued the work by interviewing a small specified group of practitioners on the challenges they faced during our joint action design research effort. Similar to the previous study, we identified three challenges categories that this time divided subsequently into two to six subcategories. The results were presented in Publication V. Worthy of remark is, however, that the categories and subcategories are different between the two studies.

4.3.1 Synthesis of the U-I Data Objectives and the Targets of Iterative SW Development Cycles

For Publication VI, we studied four different types of cycles in iterative software development methods: prototyping cycles, incremental development cycles, sprints as in e.g. Scrum, and Build-Measure-Learn (BML) cycles as in Lean Startup. More specifically, we examined the targets of these cycles to understand what kind of similarities and differences they have with each other. In the following, we will go through each of these cycles and their targets, and examine how the targets could be supported with the use objectives of U-I data. The linking between the targets, as presented in Publication VI, and the U-I data objectives are summarized in Table 4.4.

The focus of a prototyping cycle is on turning something abstract into concrete. This is done to communicate and to get feedback on the initial ideas and intuitive ideas. The feedback can be of different forms. For example, a prototype can be developed to evaluate technical aspects as well as design. Either way, feedback is easier to gather if its giver gets a concrete prototype in their hands. Occasionally, prototypes can be implemented also to assure management and other stakeholders for example about that their project is still on the right track.

U-I data can support similar targets than what we found for prototyping. This is highlighted in the need of getting feedback about some initial ideas and designs and in the capability of producing such information. Of the U-I data use objectives, *informed feature development* and *UX* targeted at producing insights for the development of a feature and its design. As long as the prototyping cycle produces applications, rather than paper prototypes, a prototype can be attached with U-I data collecting capabilities. Consequently, U-I data can then work at least as one source of feedback on the evaluation of the prototype.

Table 4.4: Linking between the targets of iterative software development cycles and objectives of U-I data use.

Cycle	Cycle Targets	U-I Data Use Objectives
Prototyping	• Figuring out what is technically doable • Validating designs and predicting large problems • Communication, assuring management and other stakeholders	• Informed feature development • UX
Incremental Development	• Provide value to the customers already during the project • Taking advantage of new technology • Assuring the stakeholders that the development is continuous and on-going	• Informed feature development
Sprints	• Responding to emerging user needs • Helping in execution and coordination of the work • Improving the ways of working • Guiding to frequent evaluations of new parts of the system	• Resourcing and prioritizing • Informed feature development • UX
Build-Measure-Learn	• Gathering justifiable evidence if profitable, scalable user needs exits • Evaluating if a hypothesized business model is feasible to satisfy the user needs • Learning by creating MVPs	• Requirements Validation

In incremental development, the motivation for a development cycle is in producing a new version of the application that includes for example a new feature. The target of such a cycle is in providing customers and users with value already during the development project. The development of increments if often split also time-wise. Consequently, the later developed increments can be implemented using newer technology that was not available for the first increments. Similar to prototyping, incremental development allows the software teams to easily communicate with different stakeholders about their project progress with concrete development phases.

U-I data can be used as a means to validate the early value producing targets of the incremental development cycles. If new increments of an application are hooked with a U-I data collection mechanism, they start producing insights of how the new features are actually used and whether they bring the intended value to their users. Such insights can be then used in determining if the development of the increment is sufficient and can be stopped. The U-I data use objective of *informed feature development* is similar to this in that the data is used for determining whether the development should be continued or not.

Sprints, as defined in Scrum, divide the development projects time-wise into periods of one to four weeks. The motivation behind such splitting is that in the beginning of each sprint the development tasks are re-prioritized. This allows the software team to respond to the emerging user needs, because they are able to start working on them already in the next Sprint. Sprints also include specified ceremonies that guide towards frequent evaluation of new parts of system.

For Sprints, using U-I data provides new means of supporting *resourcing and prioritizing* decisions. For example, quite simple statistics of feature use can assist practitioners in making prioritizing decisions of which feature's bug will be fixed in the ongoing sprint and which in the future. Similar to the feedback targets of prototyping, also Sprints can benefit of the additional U-I data's feedback mechanisms that are used in *informed feature development* and *UX*.

BML cycles, as defined in Lean Startup, are processed to validate business hypotheses. The cycle is started by building a Minimum Viable Product (MVP) - a product that fulfills the hypothetical customer need but is produced with minimum possible resources. The MVP and its success among real customers is then measured. Finally, the measured data is analyzed and the development team learns how correct their hypothesis was.

The objective of using U-I data for *requirements validation* highlights the need to gather evidence among the actual users. Although BML cycles are conceptually intended to validate business hypothesis, they are based on this same need. Actual use of either features or products assists in the validation of the related hypothesis. Similar to how BML cycles validate hypotheses about non existing businesses, requirements validation with U-I data can be done also on non-existing features. In such a case, a cycle much similar to BML can be used to create a prototype of the feature, of whose use is then measured and analyzed. All in all, the targets of the two are highly similar.

4.3.2 Challenges in the Utilization of U-I Data

We have studied the challenges of U-I data utilization in two different studies. In both cases, we have categorized the challenges into three groups. The challenge categorizations are summarized per publication in Tables 4.5 and 4.6.

Table 4.5: Categorizations of Challenges of Utilizing U-I Data in Publications IV

Acquiring Data	**Data Processing**	**Immaturity**
Collecting	Tools	Customer immaturity
Collecting Techniques	Analyzing	Process immaturity
Data quality and Quantity	Utilization	Privacy

In the first of the two studies, presented in Publication IV, we identified challenges related to *data acquiring, data processing*, and to different *immaturities*. The challenges in the data acquiring category were concerned mainly with problems and uncertainties in collecting and its tools. In addition, data quality and quantity were concerns of the respondents. For data processing, the need for better and easier to use tools was identified along with challenges related to analyzing and utilizing the collected data. Practitioners highlighted that the collected data on its own will be insufficient, and how analysis and clear utilization target is required. Finally, the category of immaturities categorized the lack of experience that challenges the utilization. The practitioners saw that the customers did not understand the value of collecting data from their users. However, they also answered that their own development processes were not ready to utilize data from the field. User-related data is often concerning in terms of privacy, and practitioners identified challenges related to this as well. Since the collecting and use of user related data is still a novel and evolving topic, the requirements on privacy by both law-makers and end-users can be changing quite often.

In the second study, presented in Publication V, we categorized the challenged into *value concerns, difficulties in U-I data utilization,* and *unsuitability of U-I data utilization in the current situation.* In challenges of value concerns, the practitioners had trouble seeing

Table 4.6: Categorizations of Challenges of Utilizing U-I Data in Publications V

Value Concerns	**Difficulties in U-I data utilization**	**Unsuitability of U-I data utilization in the current situation**
Low Value	Technical concerns and difficulties	Lack of resources
Unclear Value	Difficulties in the extraction and/or use of insights	Lack of support from organization
	High effort required for U-I data utilization	Conflicts with ways of working, methods, technical environment and/or culture
	Scalability concerns for U-I data utilization	
	Lack of experience in U-I data utilization	
	Using unspecific objectives	

the benefit of utilizing U-I data. They either thought it had low or unclear value. The difficulties in U-I data utilization were wide-ranging. We identified six subcategories to characterize them: Technical concerns and difficulties, difficulties in the extraction and/or use of insights, high effort required, scalability concerns, lack of experience, and using unspecific objectives. For the final category, we identified challenges that expressed that U-I data utilization could be unsuitable for the context in some way. This could be for example because of lack of resources or lack of support from organization. The practitioners had also faced challenges related to how U-I data utilization conflicts with either the ways of working, methods, technical environment and/or culture in their context.

4.4 Summary of Contributions per Publication

Publication I presents a study where we started our work of examining different alternatives for implementing monitoring mechanisms of user-interactions. We used DS method to create an unobtrusive analytics framework for monitoring Java applications. We designed a demonstrative example application using AOP, and integrated this collecting feature with an open-source target application available in GitHub[2] (a demo available in Vaadin's website[3]). Even if the application was a simple one, it supported the aims of the study very well. Being a fully-functional Java program, its successful integration with the developed collecting mechanism provided us with a quite easy-to-comprehend demonstration we could showcase around.

In the study presented in **Publication II**, we continued the search for different approaches to collect data from user-interactions and studied how to select the technique. Again, we used DS but supported it this time with a case study in a large international telecommunications company. We designed a framework for selecting U-I data collecting techniques. The framework includes a list of evaluation criteria for collecting techniques and a process to guide practitioners in selecting a suitable one. The framework is based on challenges organizations can face when they are starting the collection and use of automatically gathered quantitative data from user-interactions. In the publication, we

[2]https://github.com/vaadin/dashboard-demo
[3]http://demo.vaadin.com/dashboard

listed five collecting techniques, and presented an evaluation of each according to the criteria included in the framework. The empirical evaluation of the framework was left for future work.

For **Publication III**, we studied what reasons software teams have for selecting a specific technique for U-I data collecting. We conducted a multiple case study with three organizations and among other things developed an open source tool for collecting U-I data from JavaScript applications. The multiple case study was aimed at refining the selection framework from the Publication II. We found that the original list of evaluation criteria required a few adjustments because the case teams valued also reasons we had not considered in our previous study. All in all, the publication contributes in describing the processes and reasons the case teams had for selecting a suitable tool to collect U-I data.

Publication IV presents a study where we examined how software teams can utilize automatically collected PDD in software engineering. The study was conducted as a questionnaire survey where we analyzed sources, targets, and challenges that software teams can have. The survey results were collected from four Finnish software-intensive companies. The study contributes especially in categorizing the targets and in uncovering the status of the PDD use for software development in Finnish software industry.

Publication V presents our study on how software teams started the use of U-I data and what kind of challenges they faced with it. The study was conducted with ADR method and with the same multiple case setting as Publication III. In the study, we designed a utilization method for U-I data that describes the actions that the case teams took as they started the use of U-I data with us. As a result, we categorized the challenges faced by the case teams. In addition, the objective categorizations from Publication IV used as a starting point in this study were then refined and validated based on the case results. The refined objectives provide software teams with inspiration whereas the found challenges give them valuable consideration points.

Publication VI present a study on what kind of similarities and differences different types of software development iterations have with each other. The study is conceptual and in it we considered the characteristics of four archetypes of software development iterations. As a result, we identified the targets of the different iterations. The study contributes in explaining how the iterations with different goals can coexist so that one form of iteration can be used as a tool to complete the goals of another.

5 Analysis and Discussion of the Results

In this chapter we first discuss the results of the thesis to the related literature in Section 5.1. In Section 5.2 the contributions of the thesis are presented. In Section 5.3 we discuss the quality and trustworthiness of the designed artifacts. In Section 5.4 we take a look at the opportunities for future work.

5.1 Revisiting the Research Questions

RQ 1. How to select a technique for collecting user-interaction data?

Within this first research question, our aim was to study techniques for the automatic collecting of U-I data. We identified five different techniques first on a conceptual level. We then conducted case studies, where we created criteria for evaluating the techniques. Within the same studies, we developed a process that models and guides software teams' selecting of the U-I data collecting techniques. In combination with the evaluation criteria the selection process forms the framework for selecting U-I data collecting techniques.

To the best of our knowledge, this thesis is the first scientific work that has addressed the U-I data collecting techniques in this level of detail. Although many others have done research on the feedback collecting techniques, e.g. [47, 58, 59], these studies have included also other feedback collecting techniques than what work specifically for automated quantitative U-I data. The results of the thesis work satisfied the original research aims. We were able to assist software teams in practice by developing the selection framework and start collecting U-I data with them. Each of the case teams we worked with selected a different collecting technique and we were able to gather also academically valuable knowledge about how teams select the technique. However, since we were able to include only three case teams in our research, there is still room for trying out the remaining two techniques also in practice.

RQ 2. How to use user-interaction data in software development?

The aim of the second research segment was to gain an understanding of the objectives software teams have for the utilization of U-I data and to design guidance for them to start the utilization. We approached this aim from both a practical and a more theoretical standpoint. First, we conducted a questionnaire study with which we identified objectives for the analysis and use of U-I data in software teams. We then continued with a more practical approach, and conducted a multiple case study with the three case teams to gain a deeper understanding of their objectives with U-I data. Based on those case studies we refined the objective categories. In addition, we modeled the activities of the case teams by developing a utilization method for U-I data.

All of the analysis objectives found in this thesis research (*value analysis, pain point analysis, use path analysis, and user profiling*) are known and studied analysis types in the previous scientific literature as distinct analysis methods. However in the context of software engineering research, our work bundled the different options quite originally to create guidance for software teams in what kind of analysis objectives are available. On the other hand for actually conducting such analyses, we strongly recommend the Usage Mining Method by Pachidi et al. [75]. Unlike our work that concentrates on the objectives of the analyses, the Usage Mining Method considers the different analyses from the point of analysis techniques. In this sense, it can provide practitioners with additional guidance on which analysis techniques to use.

The use objectives identified in our work (*informed feature development, requirements validation, resourcing and prioritizing, and user experience*) have a lot of similarities with the utilization examples of the SOK framework's [34] development perspective. However, in addition to U-I data the SOK framework considers other data types as well. Perhaps, it is therefore able to suggest plenty of utilization examples also in other perspectives than development. Considering the increasing cross-functionality of software teams, we see that such open-mindedness can be very fruitful for practitioners collecting PDD.

Also our utilization method for U-I data has many similarities with other PDD related methods presented in the scientific literature. The method consists of three steps. The first step concentrates on the data collecting techniques and their selecting, the second on the analysis objectives, and the last one on actually collecting and using the data. The mentioned SOK framework is the most similar to our method in that it is also divided into stages of what is done to the data. However, it lacks the steps of selecting data collecting techniques. On the other hand, the Usage Mining Method by Pachidi et al. [75] considers mainly the analysis techniques and QCD [95] takes the viewpoint from the higher process level.

To summarize, our work bundled the different analysis and use objectives for U-I data and we developed a guide for how to start utilizing such data. Together with the previously presented literature, the results present categorized sets of descriptions of how to use U-I data in software development and how to start such a process in practice.

RQ 3. What kind of opportunities and challenges are there in the utilization of user-interaction data?

In the third research segment, we aimed at understanding the integration of U-I data into the processes of software teams. First, we analyzed the opportunities of using U-I data conceptually. We dissected some of the typical iterative software development methods and then analyzed how U-I data could support them. We found that many of the objectives are similar and that, at leasts on a conceptual level, U-I data can work as a very suitable feedback source in iterative software development.

Secondly, we examined the challenges that software teams face when starting the use of U-I data. We studied the challenges in two ways. We surveyed practitioners' opinions and experiences in an email questionnaire without restricting the respondents to the ones who actually had tried to use PDD in their work. However, after the multiple case study where the three case teams started the use of PDD we interviewed the teams to gather results also from practitioners with evident experiences on the topic. Therefore, the first results might reflect more the preliminary opinions of the practitioners, whereas the second ones should summarize the experiences of the practitioners on U-I data utilization.

In both of the study approaches on the challenges, we categorized the found challenges into three. With the survey, the challenges concerned the acquiring and processing of data and the overall immaturity of the topic. On the other hand, such challenges were not as highly emphasized in our second study. In that, the practitioners' answers highlighted value concerns, difficulties in U-I data utilization, and the unsuitability of U-I data utilization in their current situation. Considering the different backgrounds of the studied practitioners between the two studies, it is possible to have such a movement from the more technical challenges towards more business and organization related challenges. This can reflect also the focus of our research, which was more on solving technical problems.

The related literature on experimentation challenges has found many similar results as our work. Customer unwillingness and context challenges [12, 13, 104] were reflected in our categories of *immaturity* and *the unsuitability of U-I data utilization in the current situation*. Privacy considerations are always important, and both the related literature [101] and our work identified these as well. In addition, defining an overall evaluation metric can be tricky [94, 99, 101], and this was also seen in our studies where we found *using unspecific objectives* a challenge.

There are two challenges that we find the most concerning. Firstly, one could think that the scalability issues of qualitative feedback data [106] would be mitigated when using quantitative data. However, as the volume and complexity of data sets increase, big data related challenges [107] rise which was also reflected especially in the results of our questionnaire study. In addition, the scalability challenges are not restricted to technical considerations. Holmström-Olsson et al. [102] point out challenges also in the scalability of the impacts of experimentation. The results of our multiple case study also support this notion. Secondly, there were clearly concerns on the value that U-I data utilization brings in our multiple case study. Similar findings are categorized as lack of data insights e.g. in [101]. Mattos et al. [101] point out that in cases of unclear value, the additional cost of involving data scientists in the development might become too big of a step. In such circumstances, the experimentation could be started with some of the more basic statistical analyses to display the benefits [101].

All in all, the thesis work satisfied the aims of the third research segment. After analyzing the objectives of iterative software development methods, we found many similarities in how U-I data can support them on the conceptual level. However, our studies on the challenges also revealed that the integrations will have issues as well. Since the benefits in the conceptual level are plenty, focusing on more specific U-I data use objectives could be well-advised to gain deeper insights also in the practical level.

5.2 Contributions of the Thesis

This thesis work aimed at studying, developing, and discussing tools, techniques, methods, and processes for using U-I data in software development. The contributions have both theoretical and practical implications.

Theoretical Contributions. For academia, this thesis contributes to examining PDD, specifically regarding U-I data, and its use in software development. The main implications are the addition of new knowledge in the research field of software engineering and describing the use of U-I data both technically and on team level. Since the techniques and practices regarding automated quantitative feedback collecting are rather novel, the thesis work presents valuable results on an area where only little research exists previously.

The thesis contributed to the theoretical field from two viewpoints. As research of the software technology the thesis identified and evaluated five U-I data collecting techniques. From the software engineering research perspective we studied and modeled the methods and processes of selecting and utilizing U-I data. To add more knowledge in the research field, we categorized U-I data objectives for analysis and utilization. In addition, we presented results on how U-I data can support the targets of iterative software development cycles and what kind of challenges software teams face in practice.

Practical Contributions. For practitioners, our aim was to produce an actionable set of tools and methods that would assist them in moving towards data-driven software development with U-I data. With and during the thesis work, we created insights on how to and tools to do so. The technique selection framework was tested in practice with three software teams. For one of the cases, a concrete collecting tool was developed in JavaScript. The tool was licensed as open source and made available in GitHub. The categorizations of U-I data analysis and use objectives and their synthesis with the targets of iterative software development cycles provide practitioners with insightful ideas on where to use U-I data. In addition, the developed utilization method and the found challenges work as guidance in starting the use of U-I data in software development.

5.3 Quality Evaluation of the Designed Artifacts

The research validity for the theoretical knowledge creation was considered with the research design, but for evaluating the quality of the designed artifacts we now use the categorization by Guba [114]. To summarize, the research artifacts with practical contributions are listed in Table 5.1 along with the related artifact type and the initially used evaluation method type. For the artifact and evaluation method types, we use the distinctions by Peffers et al [115].

Table 5.1: Practical contributions and their artifact and evaluation method types in the research work.

No.	Designed Artifact	Artifact Type	Evaluation Method
1	Design of a demonstrative tool stack for unobtrusive analytics	Instantiation	Prototype
2	Selection method for U-I data collecting techniques	Method	Case Study
3	Open source tool for collecting U-I data from JavaScript applications	Instantiation	Prototype
4	Utilization method for U-I data	Method	Action Research

Credibility considers the truthfulness of the results and it can be enhanced for example by verifying the results with the study participants [116]. In this research effort, the designed artifacts 1, 2, and 4 were showcased to every participating software team. The artifact 3 was published publicly in GitHub, and a version of it was used by one of the participating software teams.

Dependability describes the constancy of the results in similar contexts [116]. Considering a situation where another research group had designed the artifacts, much of their details could have been different. However, the co-operation with the participating software teams in designing the artifacts (2-4) increases the dependability of the contributions.

Transferability refers to how the findings can be applied to other contexts [116]. In our research, convenience sampling was used and all participating software teams were Finnish. However, the teams were highly different from each other. Considering that the major common factor was the lack of experience in U-I data utilization, this should characterize also other audiences where the designed artifacts would be transferable. In addition, since the language of the artifact 3 (JavaScript) is very popular and the artifact is licensed as open source license its transferability should be high.

Confirmability should be considered for the two method-artifacts (2 & 4). It refers to how the research data represents the actual situations versus the researcher's own views [116]. In the case of artifacts 2 & 4, only one person designed the majority of them. However, the artifacts were designed in co-operation with the participating software teams' members. In addition, the artifacts were accepted to be used in each case to the degree of which they were aimed for. The designed artifacts fulfilled their aims in each of the cases.

5.4 Future Work

This thesis research brought new knowledge to how software teams use U-I data in software development. Although there is already some literature on this, the research topic is still very novel. Rodríguez et al. [13] describe in their systematic mapping study that the literature and approaches especially for customer feedback processing and its integration remains scarce. Therefore, and based on the work in this thesis, we see multiple opportunities for future research topics in this novel area.

Firstly, both the presented selection framework and the utilization method could be validated further. Their creation and initial validations were possible with the used research methods now. Three of the five identified U-I data collecting techniques were tested in these case studies, and therefore the two remaining techniques provide an obvious research topic for the future. With such a similar research setting, the similar case study method could be continued. However, in the future also different research methods for stronger evaluation should be used to mitigate the possible bias of high researcher involvement. Furthermore, the selection framework and the utilization method could then be tested with a larger set of software teams.

Secondly, new kinds of utilization objectives for U-I data could be unraveled if research methods that produce more generalizable results were used. For example, new regulations such as the European GDPR [1] offers great new sources of research data since software organizations have to describe what they do with the collected data. Perhaps even more importantly, however, describing and finding some of the common factors in U-I data collecting success stories would be important to advance the field. To us, it seems that for example the game companies who collect and use U-I data already in their daily work could provide a fruitful ground for such research. Their use of the data seems to be a degree easier in that they use it to configure features rather than to guide the whole development process. There are already some examples available, e.g. [62], but these are restricted to research with case study methods.

Finally, the found challenges require mitigation strategies in practice. Scientific literature has already proposed some [101], but actual experiences are still lacking. Since some of

[1]https://www.eugdpr.org/

the most concerning challenges were related to the low or unclear value of U-I data use and the practitioners also had difficulties in the extraction of insights, we think that the future research should focus on developing easy to use tools that generate results from the more basic analyses.

6 Conclusions

The main research problem in this thesis was to produce an actionable set of tools and methods for practitioners to start guiding their work towards data-driven software development with U-I data. Using feedback has always been an invaluable part of software development, and the use of U-I data creates a new fast-paced option for it. However, the use of U-I data in software development is not yet a wide-spread mechanism among practitioners. The selecting of a suitable data collecting technique is not an easy task, and understanding how to use the collected U-I data can be difficult as well. Since there are many opportunities and challenges related to the topic, expertise is required from software teams before they can start using this new feedback source.

The results of this thesis describe how software teams can use U-I data in software development and give them actionable means to collect the data. We conducted case- and questionnaire studies, but also intervened with the work of practitioners to actually try the using of U-I data with a hands-on approach. Firstly, the designed selection framework assists practitioners in deciding what kind of U-I data collecting techniques are available and which one would suit their needs. Our designs of the U-I data collecting tool and the demonstrative tool stack for analytics create a concrete starting point for software teams. Secondly, the four analysis and four use objectives describe the needs and insights of software teams in how to apply U-I data to their work. In addition, the three step U-I data utilization method gives software teams a clear guide for what to do to integrate the collecting of U-I data into their software system and its development. Finally, the methodological synthesis of the U-I data objectives and the objectives of iterative software development cycles explains the various opportunities of using U-I data in development work. Together with the descriptions of three challenge categories of U-I data use, the synthesis provides software practitioners and researchers with insights into U-I data in software development.

This thesis contributed in making software development more data-driven by studying and developing the novel feedback gathering mechanism of using U-I data. Being more data-driven allows software teams to base their development decisions on facts rather than on gut feelings. Using specifically U-I data focuses them on understanding their users better. Since the practices for this feedback gathering mechanism are still very much in their infancy, there is a lot of room for future work - perhaps by focusing on some of the use objectives that we found in this work. However, the main contribution of this thesis is that it generated an actionable set of tools and methods for practitioners to start the use of U-I data. Already, many of the tools designed during this thesis have been integrated into the software systems of the participated practitioners. For other software teams, the results of this thesis work as a basis and a guidance towards data-driven software development with user-interaction data.

Bibliography

[1] H. Chen, R. H. Chiang, and V. C. Storey, "Business intelligence and analytics: from big data to big impact," *MIS quarterly*, pp. 1165–1188, 2012.

[2] A. L. Montgomery, S. Li, K. Srinivasan, and J. C. Liechty, "Modeling online browsing and path analysis using clickstream data," *Marketing science*, vol. 23, no. 4, pp. 579–595, 2004.

[3] E. Lindgren and J. Münch, "Raising the odds of success: the current state of experimentation in product development," *Information and Software Technology*, vol. 77, pp. 80–91, 2016.

[4] A. E. Hassan, "The road ahead for mining software repositories," in *Frontiers of Software Maintenance, 2008. FoSM 2008.* IEEE, 2008, pp. 48–57.

[5] P. A. Brooks and A. M. Memon, "Automated gui testing guided by usage profiles," in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering.* ACM, 2007, pp. 333–342.

[6] J. Bosch, "Building products as innovation experiment systems," in *Software Business.* Springer, 2012, pp. 27–39.

[7] S. Rigatelli and S. Tuominen. (2017) Sinun tietosi eivät ole sinun – näin mainostajat keräävät ja hyödyntävät dataasi, kun muutat, ostat kilon pekonia tai haaveilet lomareissusta. [Online]. Available: https://yle.fi/uutiset/3-9901680

[8] J. Gulliksen, B. Göransson, I. Boivie, S. Blomkvist, J. Persson, and Å. Cajander, "Key principles for user-centred systems design," *Behaviour and Information Technology*, vol. 22, no. 6, pp. 397–409, 2003.

[9] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries *et al.*, "The agile manifesto," 2001.

[10] T. Dybå and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," *Information and software technology*, vol. 50, no. 9, pp. 833–859, 2008.

[11] J. Järvinen, T. Huomo, T. Mikkonen, and P. Tyrväinen, *From Agile Software Development to Mercury Business.* Cham: Springer International Publishing, 2014, pp. 58–71. [Online]. Available: https://doi.org/10.1007/978-3-319-08738-2_5

[12] M. Leppänen, S. Mäkinen, M. Pagels, V.-P. Eloranta, J. Itkonen, M. V. Mäntylä, and T. Männistö, "The highways and country roads to continuous deployment," *Software, IEEE*, vol. 32, no. 2, pp. 64–72, 2015.

[13] P. Rodriguez, A. Haghighatkhah, L. E. Lwakatare, S. Teppola, T. Suomalainen, J. Eskeli, T. Karvonen, P. Kuvaja, J. M. Verner, and M. Oivo, “Continuous deployment of software intensive products and services: A systematic mapping study,” *Journal of Systems and Software*, vol. 123, pp. 263–291, 2017.

[14] L. E. Lwakatare, “Devops adoption and implementation in software development practice. concept, practices, benefits and challenges,” Ph.D. dissertation, University of Oulu, 2017.

[15] W. E. Deming, *The new economics: for industry, government, education.* MIT press, 2000.

[16] J. Bosch, *Speed, Data, and Ecosystems: Excelling in a Software-Driven World.* CRC Press, 2017.

[17] R. H. Alan, T. M. Salvatore, P. Jinsoo, and R. Sudha, “Design science in information systems research,” *MIS quarterly*, vol. 28, no. 1, pp. 75–105, 2004.

[18] M. K. Sein, O. Henfridsson, S. Purao, M. Rossi, and R. Lindgren, “Action design research,” *MIS quarterly*, pp. 37–56, 2011.

[19] A. Kaushik, *Web Analytics 2.0.* Indianapolis, Indiana: Wiley Publishing, Inc., 2010.

[20] T. H. Davenport and J. G. Harris, *Competing on analytics: The new science of winning.* Harvard Business Press, 2007.

[21] R. P. Buse and T. Zimmermann, “Analytics for software development,” in *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, ser. FoSER ’10. New York, NY, USA: ACM, 2010, pp. 77–80. [Online]. Available: http://doi.acm.org/10.1145/1882362.1882379

[22] A. Van Barneveld, K. E. Arnold, and J. P. Campbell, “Analytics in higher education: Establishing a common language,” *EDUCAUSE learning initiative*, vol. 1, no. 1, pp. 1–ll, 2012.

[23] A. Banerjee, T. Bandyopadhyay, and P. Acharya, “Data analytics: Hyped up aspirations or true potential?” *Vikalpa*, vol. 38, no. 4, pp. 1–12, 2013.

[24] D. Gartner, Inc.: Laney. (2012) Information economics, big data and the art of the possible with analytics. [Online]. Available: https://www-01.ibm.com/events/wwe/grp/grp037.nsf/vLookupPDFs/Gartner_Doug-%20Analytics/$file/Gartner_Doug-%20Analytics.pdf

[25] D. Delen and H. Demirkan, “Data, information and analytics as services,” *Decision Support Systems*, vol. 55, no. 1, pp. 359 – 363, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167923612001558

[26] C. Holsapple, A. Lee-Post, and R. Pakath, “A unified foundation for business analytics,” *Decision Support Systems*, vol. 64, pp. 130 – 141, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167923614001730

[27] D. Zhang, S. Han, Y. Dang, J.-G. Lou, H. Zhang, and T. Xie, “Software analytics in practice,” *IEEE software*, vol. 30, no. 5, pp. 30–37, 2013.

[28] V. R. Basili, “Software modeling and measurement: the goal/question/metric paradigm,” Tech. Rep., 1992.

[29] T. H. Davenport, J. G. Harris, and R. Morison, *Analytics at work: Smarter decisions, better results.* Harvard Business Press, 2010.

[30] N. Fenton and J. Bieman, *Software metrics: a rigorous and practical approach.* CRC Press, 2014.

[31] E. Kupiainen, M. V. Mäntylä, and J. Itkonen, “Using metrics in agile and lean software development – a systematic literature review of industrial studies,” *Information and Software Technology*, vol. 62, pp. 143 – 163, 2015.

[32] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader).* Pearson Education, 2010.

[33] B. Plattner and J. Nievergelt, “Special feature: Monitoring program execution: A survey,” *Computer*, no. 11, pp. 76 – 93, 1981.

[34] H. van der Schuur, S. Jansen, and S. Brinkkemper, “A reference framework for utilization of software operation knowledge,” in *Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on.* IEEE, 2010, pp. 245 – 254.

[35] H. H. Olsson and J. Bosch, “Post-deployment data collection in software-intensive embedded products,” in *Continuous Software Engineering.* Springer, 2014, pp. 143 – 154.

[36] W. Maalej, T. Fritz, and R. Robbes, “Collecting and processing interaction data for recommendation systems,” in *Recommendation Systems in Software Engineering.* Springer, 2014, pp. 173 – 197.

[37] S. Sampath, V. Mihaylov, A. Souter, and L. Pollock, “A scalable approach to user-session based testing of web applications through concept analysis,” in *Proceedings of the 19th IEEE international conference on Automated software engineering.* IEEE Computer Society, 2004, pp. 132 – 141.

[38] S. Sprenkle, S. Sampath, E. Gibson, L. Pollock, and A. Souter, “An empirical comparison of test suite reduction techniques for user-session-based testing of web applications,” in *Software Maintenance, 2005. ICSM’05. Proceedings of the 21st IEEE International Conference on.* IEEE, 2005, pp. 587 – 596.

[39] S. Elbaum, G. Rothermel, S. Karre, and M. Fisher II, “Leveraging user-session data to support web application testing,” *IEEE Transactions on Software Engineering*, vol. 31, no. 3, pp. 187 – 202, 2005.

[40] S. Elbaum, S. Karre, and G. Rothermel, “Improving web application testing with user session data,” in *Software Engineering, 2003. Proceedings. 25th International Conference on.* IEEE, 2003, pp. 49 – 59.

[41] T. Barik, R. DeLine, S. Drucker, and D. Fisher, “The bones of the system: a case study of logging and telemetry at microsoft,” in *Proceedings of the 38th International Conference on Software Engineering Companion.* ACM, 2016, pp. 92 – 101.

[42] J. Lee, M. Podlaseck, E. Schonberg, and R. Hoch, “Visualization and analysis of clickstream data of online stores for understanding web merchandising,” *Data Mining and Knowledge Discovery*, vol. 5, no. 1-2, pp. 59−84, 2001.

[43] R. E. Bucklin and C. Sismeiro, “A model of web site browsing behavior estimated on clickstream data,” *Journal of marketing research*, vol. 40, no. 3, pp. 249−267, 2003.

[44] P. Tyrväinen, M. Saarikallio, T. Aho, T. Lehtonen, and R. Paukeri, “Metrics framework for cycle-time reduction in software value creation,” in *ICSEA 2015: The Tenth International Conference on Software Engineering Advances, ISBN 978-1-61208-438-1.* IARIA, 2015, pp. 220−227.

[45] A. Fabijan, H. H. Olsson, and J. Bosch, “Time to say’good bye’: Feature lifecycle,” in *Software Engineering and Advanced Applications (SEAA), 2016 42th Euromicro Conference on.* IEEE, 2016, pp. 9−16.

[46] S. Marciuska, C. Gencel, and P. Abrahamsson, “Exploring how feature usage relates to customer perceived value: A case study in a startup company,” in *International Conference of Software Business.* Springer, 2013, pp. 166−177.

[47] S. Marciuska, C. Gencel, X. Wang, and P. Abrahamsson, “Feature usage diagram for feature reduction,” in *International Conference on Agile Software Development.* Springer, 2013, pp. 223−237.

[48] J. Hemilä, J. Vilko, E. Kallionpää, and J. Rantala, “Value creation in product-service supply networks,” in *The Proceedings of 19th International Symposium on Logistics (ISL 2014), 6.-9.7. 2014., Ho Chi Minh City, Vietnam*, 2014, pp. 168−175.

[49] H. H. Olsson, H. Alahyari, and J. Bosch, “Climbing the" stairway to heaven"−a mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software,” in *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on.* IEEE, 2012, pp. 392−399.

[50] A. Begel, Y. P. Khoo, and T. Zimmermann, “Codebook: discovering and exploiting relationships in software repositories,” in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1.* ACM, 2010, pp. 125−134.

[51] A. J. Ko, R. DeLine, and G. Venolia, “Information needs in collocated software development teams,” in *Software Engineering, 2007. ICSE 2007. 29th International Conference on.* IEEE, 2007, pp. 344−353.

[52] J. Sillito, G. C. Murphy, and K. De Volder, “Asking and answering questions during a programming change task,” *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 434−451, 2008.

[53] A. Begel and T. Zimmermann, “Analyze this! 145 questions for data scientists in software engineering,” in *Proceedings of the 36th International Conference on Software Engineering.* ACM, 2014, pp. 12−23.

[54] R. P. Buse and T. Zimmermann, “Information needs for software development analytics,” in *Proceedings of the 34th international conference on software engineering.* IEEE Press, 2012, pp. 987−996.

[55] E. Backlund, M. Bolle, M. Tichy, H. H. Olsson, and J. Bosch, “Automated user interaction analysis for workflow-based web portals,” in *Software Business. Towards Continuous Value Delivery.* Springer, 2014, pp. 148–162.

[56] D. I. Mattos, J. Bosch, and H. H. Olsson, “More for less: Automated experimentation in software-intensive systems,” in *International Conference on Product-Focused Software Process Improvement.* Springer, 2017, pp. 146–161.

[57] R. Junco, “Comparing actual and self-reported measures of facebook use,” *Computers in Human Behavior*, vol. 29, no. 3, pp. 626–631, 2013.

[58] A. Holzinger, “Usability engineering methods for software developers,” *Communications of the ACM*, vol. 48, no. 1, pp. 71–74, 2005.

[59] A. Fabijan, H. H. Olsson, and J. Bosch, “Customer feedback and data collection techniques in software r&d: a literature review,” in *Software Business.* Springer, 2015, pp. 139–153.

[60] T. Sauvola, L. E. Lwakatare, T. Karvonen, P. Kuvaja, H. H. Olsson, J. Bosch, and M. Oivo, “Towards customer-centric software development: A multiple-case study,” in *Software Engineering and Advanced Applications (SEAA), 2015 41st Euromicro Conference on.* IEEE, 2015, pp. 9–17.

[61] T. Roehm, B. Bruegge, T.-M. Hesse, and B. Paech, “Towards identification of software improvements and specification updates by comparing monitored and specified end-user behavior,” in *Software Maintenance (ICSM), 2013 29th IEEE International Conference on.* IEEE, 2013, pp. 464–467.

[62] J. H. Kim, D. V. Gunn, E. Schuh, B. Phillips, R. J. Pagulayan, and D. Wixon, “Tracking real-time user experience (true): a comprehensive instrumentation solution for complex systems,” in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems.* ACM, 2008, pp. 443–452.

[63] S. Westerman, S. Hambly, C. Alder, C. Wyatt-Millington, N. Shryane, C. Crawshaw, and G. Hockey, “Investigating the human-computer interface using the datalogger,” *Behavior Research Methods, Instruments, & Computers*, vol. 28, no. 4, pp. 603–606, 1996.

[64] S. Trewin, “Inputlogger: General-purpose logging of keyboard and mouse events on an apple macintosh,” *Behavior Research Methods, Instruments, & Computers*, vol. 30, no. 2, pp. 327–331, 1998.

[65] U. Kukreja, W. E. Stevenson, and F. E. Ritter, “Rui: Recording user input from interfaces under windows and mac os x,” *Behavior Research Methods*, vol. 38, no. 4, pp. 656–659, 2006.

[66] J. Alexander, A. Cockburn, and R. Lobb, “Appmonitor: A tool for recording user actions in unmodified windows applications,” *Behavior Research Methods*, vol. 40, no. 2, pp. 413–421, 2008.

[67] J. Matejka, T. Grossman, and G. Fitzmaurice, “Patina: Dynamic heatmaps for visualizing application usage,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* ACM, 2013, pp. 3227–3236.

[68] J. P. Magalhaes and L. M. Silva, "Anomaly detection techniques for web-based applications: An experimental study," in *Network Computing and Applications (NCA), 2012 11th IEEE International Symposium on*. IEEE, 2012, pp. 181–190.

[69] R. Musson, J. Richards, D. Fisher, C. Bird, B. Bussone, and S. Ganguly, "Leveraging the crowd: how 48,000 users helped improve lync performance," *IEEE software*, vol. 30, no. 4, pp. 38–45, 2013.

[70] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-oriented programming," *ECOOP'97—Object-oriented programming*, pp. 220–242, 1997.

[71] E. C. Groen and M. Koch. (2016) How requirements engineering can benefit from crowds. [Online]. Available: https://re-magazine.ireb.org/articles/how-requirements-engineering-can-benefit-from-crowds

[72] S. H. Kan, *Metrics and models in software quality engineering*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[73] M. El-Ramly, E. Stroulia, and P. Sorenson, "Recovering software requirements from system-user interaction traces," in *Proceedings of the 14th international conference on Software engineering and knowledge engineering*. ACM, 2002, pp. 447–454.

[74] M. El-Ramly and E. Stroulia, "Mining software usage data," in *Proceedings of 1st International Workshop on Mining Software Repositories (MSR'04)*, 2004, pp. 64–68.

[75] S. Pachidi, M. Spruit, and I. Van De Weerd, "Understanding users' behavior with software operation data mining," *Computers in Human Behavior*, vol. 30, pp. 583–594, 2014.

[76] A. Field, *Discovering statistics using SPSS*. Sage publications, 2009.

[77] L. Breiman, *Classification and regression trees*. Routledge, 2017.

[78] P. Giudici, *Applied data mining: Statistical methods for business and industry*. John Wiley & Sons, 2005.

[79] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009.

[80] T. Kohonen, "Self-organizing maps, springer series in information sciences," *Berlin: Springer*, vol. 3, p. 30, 2001.

[81] N. R. Mabroukeh and C. I. Ezeife, "A taxonomy of sequential pattern mining algorithms," *ACM Comput. Surv.*, vol. 43, no. 1, pp. 3:1–3:41, Dec. 2010. [Online]. Available: http://doi.acm.org/10.1145/1824795.1824798

[82] R. G. Cowell, P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter, *Probabilistic networks and expert systems: Exact computational methods for Bayesian networks*. Springer Science & Business Media, 2006.

[83] C. M. Grinstead and J. L. Snell, *Introduction to probability*. American Mathematical Soc., 2012.

[84] H. Kagdi, M. L. Collard, and J. I. Maletic, “A survey and taxonomy of approaches for mining software repositories in the context of software evolution,” *Journal of Software: Evolution and Process*, vol. 19, no. 2, pp. 77–131, 2007.

[85] O. Baysal, R. Holmes, and M. W. Godfrey, “Mining usage data and development artifacts,” in *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*. IEEE Press, 2012, pp. 98–107.

[86] A.-L. Mattila, T. Lehtonen, H. Terho, T. Mikkonen, and K. Systä, “Mashing up software issue management, development, and usage data,” in *Proceedings of the Second International Workshop on Rapid Continuous Software Engineering*. IEEE Press, 2015, pp. 26–29.

[87] S. Diehl, *Software visualization: visualizing the structure, behaviour, and evolution of software*. Springer Science & Business Media, 2007.

[88] H. Maylor, “Beyond the gantt chart:: Project management moving on,” *European Management Journal*, vol. 19, no. 1, pp. 92–100, 2001.

[89] M. Blom, “Is scrum and xp suitable for cse development?” *Procedia Computer Science*, vol. 1, no. 1, pp. 1511 – 1517, 2010, iCCS 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050910001699

[90] M. D’Ambros, M. Lanza, and M. Pinzger, “" a bug’s life" visualizing a bug database,” in *Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on*. IEEE, 2007, pp. 113–120.

[91] K. E. Isaacs, A. Giménez, I. Jusufi, T. Gamblin, A. Bhatele, M. Schulz, B. Hamann, and P.-T. Bremer, “State of the art of performance visualization,” *EuroVis 2014*, 2014.

[92] O. Benomar, H. Sahraoui, and P. Poulin, “A unified framework for the comprehension of software’s time,” in *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, vol. 2. IEEE, 2015, pp. 603–606.

[93] A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, “The benefits of controlled experimentation at scale,” in *Software Engineering and Advanced Applications (SEAA), 2017 43rd Euromicro Conference on*. IEEE, 2017, pp. 18–26.

[94] R. Kohavi, R. Longbotham, D. Sommerfield, and R. M. Henne, “Controlled experiments on the web: survey and practical guide,” *Data mining and knowledge discovery*, vol. 18, no. 1, pp. 140–181, 2009.

[95] H. H. Olsson, J. Bosch, and H. Alahyari, “Towards r&d as innovation experiment systems: A framework for moving beyond agile software development,” in *Proceedings of the IASTED*, 2013, pp. 798–805.

[96] F. Fagerholm, A. S. Guinea, H. Mäenpää, and J. Münch, “The right model for continuous experimentation,” *Journal of Systems and Software*, vol. 123, pp. 292–305, 2017.

[97] H. H. Olsson and J. Bosch, “From opinions to data-driven software r&d: A multi-case study on how to close the’open loop’problem,” in *Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on*. IEEE, 2014, pp. 9–16.

[98] L. Guzmán, M. Oriol, P. Rodríguez, X. Franch, A. Jedlitschka, and M. Oivo, "How can quality awareness support rapid software development?–a research preview," in *International Working Conference on Requirements Engineering: Foundation for Software Quality.* Springer, 2017, pp. 167–173.

[99] A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, "The evolution of continuous experimentation in software product development," in *International Conference on Software Engineering (ICSE)*, 2017, pp. 770–780.

[100] T. Karvonen, L. E. Lwakatare, T. Sauvola, J. Bosch, H. H. Olsson, P. Kuvaja, and M. Oivo, "Hitting the target: practices for moving toward innovation experiment systems," in *International Conference of Software Business.* Springer, 2015, pp. 117–131.

[101] D. I. Mattos, J. Bosch, and H. H. Olsson, "Challenges and strategies for undertaking continuous experimentation to embedded systems: Industry and research perspectives," in *International Conference on Agile Software Development.* Springer, 2018, pp. 277–292.

[102] H. H. Olsson, J. Bosch, and A. Fabijan, "Experimentation that matters: A multi-case study on the challenges with a/b testing," in *International Conference of Software Business.* Springer, 2017, pp. 179–185.

[103] J. Bosch and U. Eklund, "Eternal embedded software: Towards innovation experiment systems," in *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation.* Springer, 2012, pp. 19–31.

[104] G. G. Claps, R. B. Svensson, and A. Aurum, "On the journey to continuous deployment: Technical and social challenges along the way," *Information and Software technology*, vol. 57, pp. 21–31, 2015.

[105] J. O. Johanssen, A. Kleebaum, B. Bruegge, and B. Paech, "Towards a systematic approach to integrate usage and decision knowledge in continuous software engineering." in *CSE@ SE*, 2017, pp. 7–11.

[106] T. Johann and W. Maalej, "Democratic mass participation of users in requirements engineering?" in *Requirements Engineering Conference (RE), 2015 IEEE 23rd International.* IEEE, 2015, pp. 256–261.

[107] U. Sivarajah, M. M. Kamal, Z. Irani, and V. Weerakkody, "Critical analysis of big data challenges and analytical methods," *Journal of Business Research*, vol. 70, pp. 263 – 286, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S014829631630488X

[108] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of management information systems*, vol. 24, no. 3, pp. 45–77, 2007.

[109] R. J. Wieringa, *Design science methodology for information systems and software engineering.* Springer, 2014.

[110] J. Järvinen, R. Kuusela, T. Mikkonen, and A. Turunen. (2017) Dimecc n4s-program: Finnish software companies speeding digital economy. [Online]. Available: http://n4s.dimecc.com/en/

[111] R. K. Yin, *Case study research: Design and methods.* Sage publications, 2013.

[112] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical software engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[113] H.-F. Hsieh and S. E. Shannon, "Three approaches to qualitative content analysis," *Qualitative health research*, vol. 15, no. 9, pp. 1277–1288, 2005.

[114] E. G. Guba, "Criteria for assessing the trustworthiness of naturalistic inquiries," *ECTJ*, vol. 29, no. 2, pp. 75–91, 1981.

[115] K. Peffers, M. Rothenberger, T. Tuunanen, and R. Vaezi, "Design science research evaluation," in *International Conference on Design Science Research in Information Systems.* Springer, 2012, pp. 398–410.

[116] D. G. Cope, "Methods and meanings: credibility and trustworthiness of qualitative research." in *Oncology nursing forum*, vol. 41, no. 1, 2014.

Publications

PUBLICATION
I

Designing an Unobtrusive Analytics Framework for Monitoring Java Applications

Suonsyrjä, S., Mikkonen, T.

International Workshop on Software Measurement (IWSM), pp. 160-175 Jan. 2015.

Designing an Unobtrusive Analytics Framework for Monitoring Java Applications

Sampo Suonsyrjä and Tommi Mikkonen

Department of Pervasive Computing, Tampere University of Technology,
Korkeakoulunkatu 10, FI-33720, Tampere, Finland
sampo.suonsyrja@tut.fi, tommi.mikkonen@tut.fi

Abstract. In software development, attention has recently been placed on understanding users and their interactions with systems. User studies, practices such as A/B testing, and frameworks such as Google Analytics that gather data on production use have become common approaches in particular in the context of the Web, where it is easy to perform frequent updates as new needs emerge. However, when considering installable desktop applications, the situation gets more complex. While analytics facilities are still needed, they should address business logic, not generic traffic as is the case with many web sites. Moreover, analytics should be unobtrusive, and not have a high impact on the evolution of the actual application; thus, analytics should be treated as an add-on, as the target system may already exist. Finally, the instrumentation of features that are observed should be easy and flexible, but the provided mechanisms should be expressive enough for many use cases. In this paper, we examine different alternatives for implementing such monitoring mechanisms, and report results from an experiment with Vaadin, a web framework based on Java and Google Web Toolkit, GWT.

1 Introduction

The introduction of Agile methods [6] caused a paradigm shift in the development of software systems: instead of starting with a set of requirements that are all of the same value, software developers began to embrace a model where systems are first built with only a set of key features to be later extended into a more complete form. As more and more experience regarding the use of the system is gathered, developers write new versions of the system which satisfy user needs better. In fact, one can even claim that the core of iterative development is the ability to learn in each increment, which leads to improved products.

In the process of creating the software in the above fashion, input from users of the system can play a crucial role, given that adequate mechanisms for collecting the input are available. The most traditional way is to design questionnaires or other studies that the end users answer to guide the development, but in particular in the field of web systems, also more sophisticated forms of gathering information exist regarding users and the way the system is being used. For instance A/B testing, where different sets of users use a slightly different version of the software, helps in deciding between two ways to provide similar or the same features. Moreover, analytics frameworks such as Google Analytics provide detailed understanding regarding how users interact with the system to perform more complicated tasks. In general, the ability to gather all this information is opening new possibilities for developers, because even the slightest deviations in user behavior can be tracked and reacted upon.

Although the field of web systems can nowadays be seen to have an edge in collecting post-deployment data, the same need is increasing in other contexts as well, as evidenced by [8]. In this paper, we investigate techniques for monitoring application-level user activity, as well as an option to extend the techniques to cover installable desktop applications, too.

The goal is to track actions at the level of user interface widgets, such as buttons, sliders, and text fields for instance. The work is based on using Java web framework Vaadin [5], where applications are first composed with Java, and then compiled into a form that can be deployed to the web, with the parts of the application that form the user interface being compiled with Google Web Toolkit [13]. As the concrete implementation mechanism for introducing analytics facilities, we experiment using aspect-oriented techniques [4] to bind an existing design to an external data analytics framework.

The rest of this paper is structured as follows. In Section 2, we introduce motivation and background of the study. In Section 3, we introduce our research questions. In Section 4, we describe our demonstrator application and how it has been constructed. In section 5, we provide details of our implementation: showing how data is gathered in an unobtrusive fashion and describing the design of our analytics framework. In Section 6, we provide an extended discussion regarding our findings. Finally, in Section 7, we draw final conclusions.

2 Background

Analytics is used by businesses of all type to better understand customers. During the recent years, also software engineers and software engineering organizations have understood the opportunity to use more data for making constantly better decisions, but as even sporting teams have improved their performance with the help of analytics, the uses for analytics seem to be fairly general [1].

2.1 Software Analytics

Pachidi et al. [12] have developed the Usage Mining Method that enables conducting classification analyses, user profilings and clickstream analyses on logged operation data. Such data is beneficial for program understanding and reengineering [3]. In addition, as the size and complexity of software systems continue to grow, decision making is becoming even more difficult in the future and thus new solutions such as the use of analytics data are needed [2].

Kristjansson and van der Schuur have formulated the concept Software Operation Knowledge [9]. They describe that to consist of knowledge of in-the-field performance, quality and usage of software, and knowledge of end-user experience and end-user feedback. The researchers continue with stating how software vendors have a great interest in acquiring such knowledge, but that the systematic practice of gathering, analyzing and acting on such knowledge is still limited. Correspondingly, this kind of in-the-field knowledge could benefit usability studies as the lack of long-term data collection is considered as one of the challenges in measuring usability [7].

In general, it is possible to collect usage metrics by executing software applications, but this usually requires some sort of modifications to the source code of the target application. There are a few exceptions however. For example, the Patina system [10] uses Microsoft Active Accessibility API to collect accessibility data, and thus no altering of the source code is needed. The system creates a so-called heatmap, which visualizes the content and location of the user interface controls visible in the application. As a drawback, supporting the accessibility API usually requires some extra work from the application developers and so the coverage of the accessibility API can vary.

As for concrete implementations, one of the most commonly used analysis frameworks is Google Analytics (http://www.google.com/analytics/), which is presently being used by an increasing number of web sites. With it, the developers of a web site can track traffic of a monitored web site and view it in a form that is easy to interpret. The data provides information regarding visitors, their geographical locations, the time they remain on the site, what is the path that users take on the web site, and so on. Since the system operates in the Web, its operation can rely on web protocols that reveal these properties. For a generic desktop application, however, these facilities are not immediately available. Moreover, when considering installable applications, data to be collected is often application specific, not web traffic related as is the case with Google Analytics. However, the popularity of Google Analytics demonstrates that there is an increasing interest regarding user data, which can be made available in an unobtrusive fashion.

2.2 Aspect-Oriented Programming

Aspect-oriented software development provides means for capturing cross-cutting concerns and modularizing them as manageable units [4]. Tackling the issue of tangled code, aspect-oriented programming languages such as AspectJ provide means to insert additional operations to a target program in an unobtrusive fashion with a new construct, so-called *aspect*. Aspects in turn provide increased opportunities for advanced modularity.

At the implementation level, an AspectJ aspect always includes at least two parts: a pointcut and an advice, both of which are code snippets. The pointcut is used to describe the point where the execution of the target program is paused for inserting the additional code programmed in the advice part. Figure 1 provides a simple aspect code that introduces a simple logging facility that records the parameters and the return value of a method call. In this aspect, the pointcut is defined to take effect around the defined function of our example class, `MyClass::MyFunc`. The `Logger` aspect takes effect as the function is called, and the aspect code is executed both before and after actually executing the original method in a fashion where its execution is not affected. The operations that are being executed before and after running the method can be arbitrary; however for the purposes of software analytics, these include data collection operations.

```
aspect Logger {
pointcut loggedFunction = call("void MyClass::MyFunc(...)");
advice loggedFunction:around() {
   // Log call and method parameters
   tjp->Proceed(); // Run MyClass::MyFunc
   // Log results
   }
}
```

Fig. 1. A sample aspect.

3 Research Questions

The research questions we formed to evaluate our usage data collection and analysis framework are the following.

RQ1: To what extent can a data collecting feature be implemented without compromising the evolution of the target program? As a starting point for our research, we have taken a view where the design and evolution of the target system, in other words the program from which usage data is to be collected, must remain as independent from data collection and analysis as possible. High priority of this independence is motivated by the fact that in the end analytics data leads to changes in the target program. Therefore, it is crucial that the target program can be under constant change and these data can still be collected from it. This leads to the selection of implementation techniques that are as unobtrusive as possible.

As the evolution of the target program results in data being collected from different versions of the target program, the approach used for collecting data has to ensure that these data are still comparable between the different versions. Thus, not only do we want to find out specific types of data that can be collected with our framework, but also if the data is adequate enough to be compared between different versions of the target program. Finally, as we aim at designing a data collection framework that is independent of the underlying target program, we also introduce an option to reuse the development effort invested in the framework in different setups, including desktop applications as well as web systems built using Java.

RQ2: What types of data can be collected with the given approach? As with any technology, there are restrictions regarding the data that can be collected. In this paper, we are interested in interactions between the user and the application, and therefore we focus on data that is associated with user interactions only. Thus, interactions with e.g. external actors or machines are beyond our scope in this paper.

RQ3: How to connect the data collecting feature with an analysis framework? Being able to record data from a user interface is only a beginning in the way towards understanding how an application is being used. Therefore, it is necessary to load the resulting usage data to an analysis system, which can then be used to further process the data into a meaningful form.

4 Demonstrator Application

To answer the above research questions, we next describe a demonstrator application. First, we introduce the platform on top of which the system is built. Then, we describe the application. Finally, we show how manual instrumentation could be carried out for this application.

4.1 Vaadin Web Framework

Vaadin [5] is an open source framework that is used for developing Rich Internet Applications (RIA). Vaadin applications are written using Java, and they are transformed into AJAX applications with the facilities of Google Web Toolkit (GWT) [13]. The architecture of the system is illustrated in Figure 2.

Vaadin applications are implemented similarly to Java Standard Edition desktop applications, with all the functionality written using Java. However, instead of using the usual Java UI libraries like AWT, SWT, or Swing, a specific set of Vaadin UI components is used. These components can be compiled into a form that is runnable inside the browser, following the development process of GWT. This process is illustrated in Figure 3. In addition, new custom made UI components can be implemented when needed to create systems with different kinds of look-and-feel.

4.2 Demonstration Application

To evaluate the designed framework for usage data collection, we selected a Vaadin application, which is fully functional and already developed yet simple enough to be the first test application. The source code is available for download at https://github.com/vaadin/dashboard-demo, and a working demo is located at http://demo.vaadin.com/dashboard.

The target application, called *QuickTickets Dashboard Demo*, demonstrates how the Vaadin framework can be used to create a simple dashboard web application. The main dashboard view is initialized as an object of DashboardView class. During the initialization, several objects of HorizontalLayout class

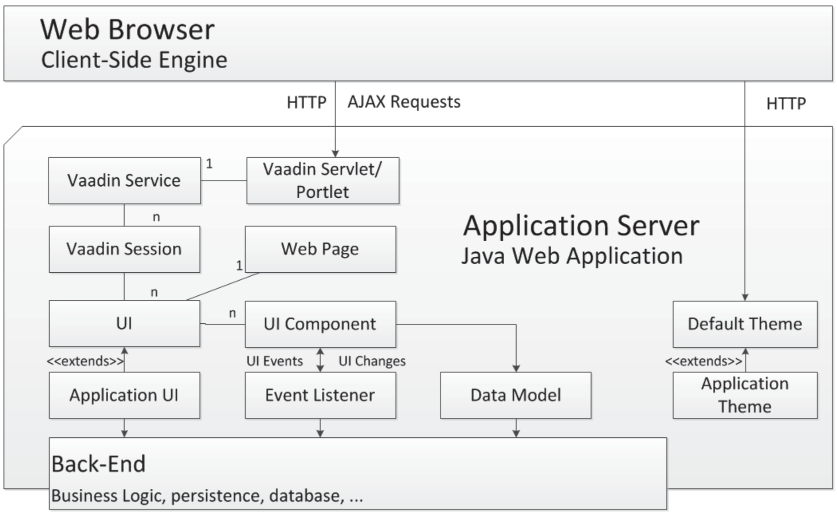


Fig. 2. Vaadin architecture. Image adapted from [5]

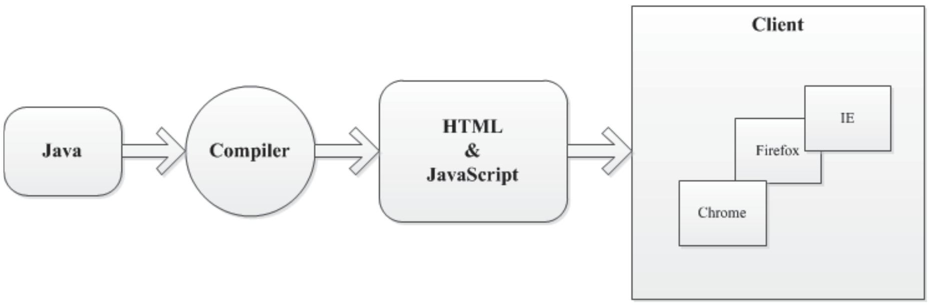


Fig. 3. GWT process of compiling Java to HTML and JavaScript [13]

are instantiated and pushed to the view with an addComponent method. These include components such as a toolbar and several rows. Buttons are added correspondingly to these layout components in the same manner. In Figure 4, we demonstrate the initialization of a dashboard object on code level along with a toolbar (a `horizontalLayout` object) and a notify button.

In the following, we demonstrate how collecting usage data works by focusing on buttons that can be pressed by users. While this obviously does not cover all the dimensions of software operation knowledge, this restriction simplifies the presentation to a form that is concrete enough to demonstrate at a detailed level how data collection works.

```
public DashboardView() {
   HorizontalLayout top = new HorizontalLayout();
   addComponent(top);
   Button notify = new Button('2');
   Notify.addClickListener(
      new ClickListener(){
         ...
      });
   top.addComponent(notify);
};
```

Fig. 4. Initialization of a dashboard object.

4.3 Manual method as a motivation

To show how the proposed automatic data collection feature simpifies developers' tasks, we first provide a manual implementation of the same function. To this end, we inserted data collecting features manually ourselves to specific places in the original source code of the target application. Thus, this approach is an intrusive one as it essentially changes the source code of the target application, which is built by someone else.

First, we developed a class called `DataLogger.java`. This class was used for two important tasks. On one hand, it included public method `logButtonClick` and on the other hand it stored these button clicks to a SQL type of a container. `Button` and `ClickEvent` objects were used as parameters for the method. It draws information about the button and its context and then stores it to the aforementioned temporary container. This information could be of course stored in some other way as well, but for our study case this was not seen important. However, some storing options are discussed in the future work section. This class itself was then included in the same java package with the target applications source code files. Up until this point the target applications source code was not altered.

In the unobtrusive part, the whole source code of the target application was then searched through to find each and every place where a new button was instantiated and added to the UI as seen in Figure 4. As with every button there was also an instantiation of its `ClickListener`, we always inserted a call to our `logEvent` method within this instantiation. In Figure 5, we provide a code snippet that elaborates how this implementation was done.

Clearly, we only used one intrusive insertion to the application, the call to method `Logger.logEvent`. However, even with this simple application, there were a total of 33 of this kind of button instantiations in the target application, all of which had to be extended with a similar call to our data logging method. While 33 insertions can be implemented once quite fast, the devil is in the complexity that most likely starts to build up when such implementation process is repeated for a while. Especially in a case where the target application is developed by a different person than the one implementing the usage data logging features, there is always the risk of forgetting to add these logging features to all the necessary

```
final Button signin = new Button("Sign In");
signin.addClickListener(
   new ClickListener() {
      public void buttonClick(ClickEvent e) {
         Logger.logEvent(signinEvent, e);
         ...
      }
   }
);
```

Fig. 5. Manual implementation of data collection.

places. Furthermore, even if a special script was developed to insert the logging features automatically to specific places, one would have to be very careful in developing such a script. Although this should reduce the risk of forgetting to log a button at all, any possible extra calls to the data logging methods would then again distort the data and its reliability as button clicks could be recorded not just once but twice or trice and so on.

In the regard of data comparability the manual approach, qualities depend greatly on the specific implementation. In this case study, our implementation gathered data only straight from the context of the target application. This included data types such as buttons caption, session id, and URI fragment. Although having all the data coming from the Vaadin frameworks context creates quite a reliable starting point for a further usage data analysis, target application evolution and changes in for example buttons captions might lead to inconsistencies in collected usage data.

In what comes to the flexibility of the manual approach and usefulness of the data it collects, we saw this approach performing understandably well. Making the application log new kinds of data types was as easy as making it log the first types of data. Of course in a case with a larger-scale application this might take more than a blink of an eye. However, the point in the flexibility criterion is to evaluate if the approach is able to collect also new kinds of data and the manual approach certainly has that as an advantage. Similarly, it collects just the types of data one wants and thus these data should be as useful as any.

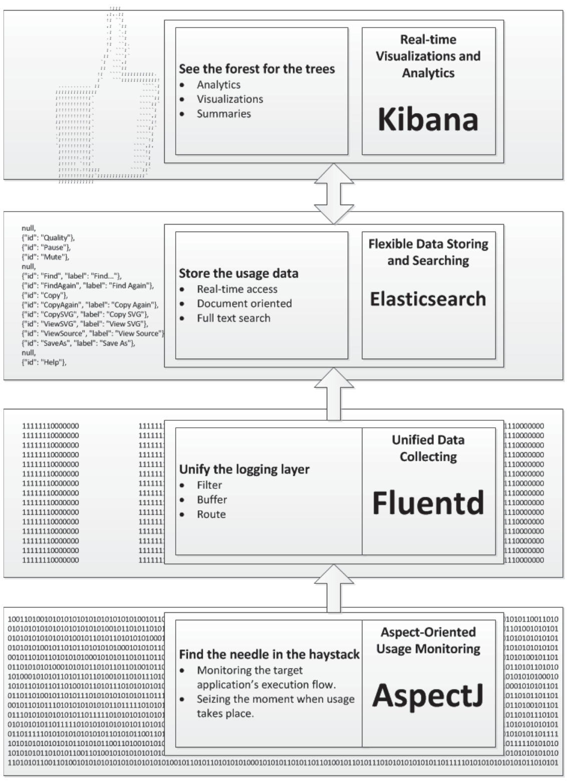
5 Data Collection and Analysis Framework

To support usage data collection, we designed a framework where several already existing techniques and tools are used (Figure 6). These key components are:

- AspectJ is used for creating an unobtrusive monitoring mechanism for the target application.
- Fluentd (www.fluentd.org) is used as the mechanism for unified data collection.
- Elasticsearch (www.elasticsearch.org/overview/elasticsearch) is used as a real-time storage for flexible searches.

– Kibana (www.elasticsearch.org/overview/kibana) is used for creating real-time visualizations and analytics.

This stack that combines Fluentd, Elasticsearch, and Kibana can be considered as an open source alternative to Splunk (www.splunk.com) log management software.



See the forest for the trees
Analytics
Visualizations
Summaries
Real-time Visualizations and Analytics
Kibana
Store the usage data
Real-time access
Document oriented
Full text search
Flexible Data Storing and Searching
Elasticsearch
Unify the logging layer
Filter
Buffer
Route
Unified Data Collecting
Fluentd
Find the needle in the haystack
Monitoring the target application's execution flow.
Seizing the moment when usage takes place.
Aspect-Oriented Usage Monitoring
AspectJ

Fig. 6. The Designed Framework for Unobtrusive Analytics.

5.1 Aspect-Oriented Usage Monitoring

The aspect-oriented approach to inserting additional features into existing applications is unobtrusive by nature. As already mentioned, we demonstrate this facility by focusing on buttons. To this end, we wish to intervene in the execution every time a button is being added to a UI component (see Figure 4 in Subsection 4.2). To attach a pointcut and a logging advice to such call, aspect `AddComponentListener` was created as shown in Figure 7.

In this aspect, pointcut called `addComponentCall` defines that each time method `addComponent` is called with a button as its parameter, the execution can be cut

```
public aspect AddComponentListener {
   // Button clicks are stored in this container.
   DataLogger dataCollector = new DataLogger();
   // To be executed when a button is added to the layout.
   pointcut addComponentCall(Button b):
      call(* *.addComponent(*)) && args(bb);
   // To be executed after a button has been added to layout.
   after(final Button b):
      addComponentCall(b) {
      // Clicks are listened to with a basic Vaadin ClickListener.
      b.addClickListener(
         new Button.ClickListener() {
         public void click(ClickEvent e) {
            dataCollector.logEvent(b, e);
         }
      });
   }
}
```

Fig. 7. Data collector aspect, its pointcut and advice.

for the corresponding advice part. This part will then define an additional click listener. This is shown in Figure 8.

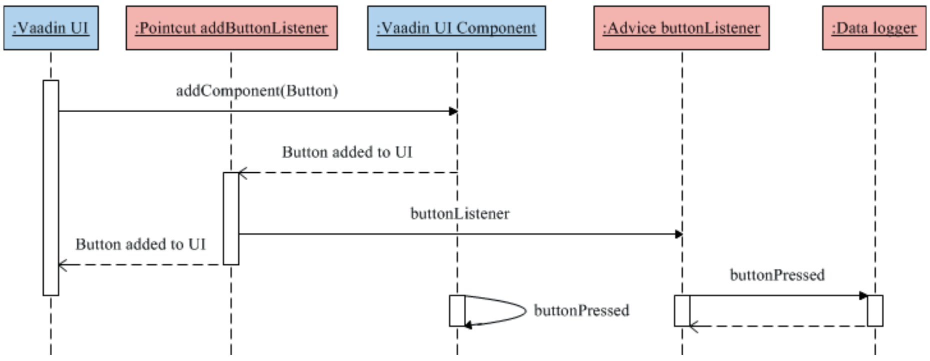


Fig. 8. Insertion of an additional click listener with an aspect.

Finally, a remark must be made regarding the degree of unobtrusiveness of the approach. While the effect of AspectJ code is unobtrusive to the underlying target program, tooling is affected by AspectJ. To begin with, for the build process, a dependency to AspectJ must be inserted to the target application's project file. Additionally, the AspectJ tools must be included in the used IDE, in our case Eclipse.

5.2 Collecting Data with Fluentd

Fluentd was implemented in quite a similar fashion as the AspectJ for monitoring features. However, wherein AspectJ was used for unobtrusively monitoring the usage, Fluentd was used for collecting usage data from the usage points defined with AspectJ. Thus, the core idea of Fluentd is to be the unifying layer between different types of log inputs and outputs. This is illustrated in Figure 9, in which a box is a component and the arrows describe the data flow.

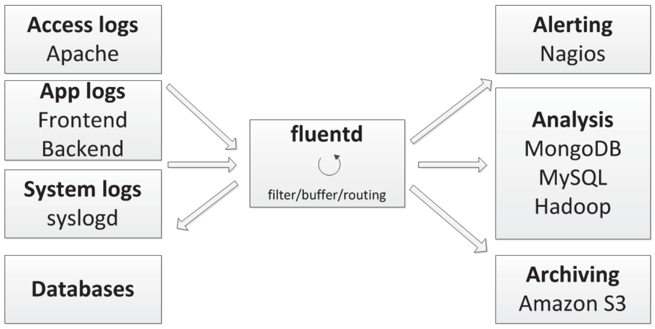


Fig. 9. Architecture of Fluentd and its plugins. Image adapted from [fluentd.org/architecture]

Figure 9 illustrates the architecture of Fluentd. Its various plugins for data input make it easier to unify the logging layer of an application or even an application ecosystem. There are a number of different input plugins available for several programming languages. In this study, we obviously used an input plugin for Java applications. However, Fluentd supports inputs not only from different language applications but also from entirely different kinds of inputs. These include for example access and error logs from web servers and system logs.

The concrete implementation of Fluentd into the target application required that a Fluentd dependency was inserted into the source code of the target application. This was done similarly as with the AspectJ facilities. Additionally, we installed and ran Fluentd on the same machine with the target application. As these requirements are met, the Fluentd process is able to receive the inputs described in Figure 10. As described with the usage monitoring aspect in Figure 7, `logButtonClick` method is called whenever a button is clicked.

Similar to the input plugins of Fluentd, its plugins for storing data standardize that front. Depending on the use case, data can be stored in different formats for archiving and analysis, for example. In this study, we used Fluentd for parsing the usage data into JSON and then forwarding them for analysis in Elasticsearch. As seen in Figure 10, there were different types of usage data related to a button click, its context, and the button itself. These data were first

```
public class DataLogger {
    private static FluentLogger LOG =
       FluentLogger.getLogger("button.click");
    public void logButtonClick(Button b, ClickEvent event){
       Map<String, Object> data = new HashMap<String, Object>();
       data.put("Uri Fragment", Page.getCurrent().getUriFragment());
       data.put("Page", Page.getCurrent().toString());
       data.put("Button Caption", b.getCaption());
       data.put("Button ID", b.getId());
       ...
       data.put("Click X", event.getClientX());
       data.put("Click Y", event.getClientY());
       LOG.log("click", data);
   }
}
```

Fig. 10. Collecting data from a Java application with Fluentd.

stored in a temporary Java Hashmap object but then forwarded to Fluentd for its filtering, buffering, and rerouting processes.

5.3 Elasticsearch and Kibana

In our study setup, we used Fluentd and Elasticsearch on the same localhost. Fluentd sent the collected usage data to Elasticsearch, which stored them into its document oriented database without any pre-configurations. As the data was already formatted in JSON, the field names were already there. This in combination with the full-text search abilities made analyzing facilities easily accessible. In addition, Elasticsearch supports real-time access to exploring the stored data.

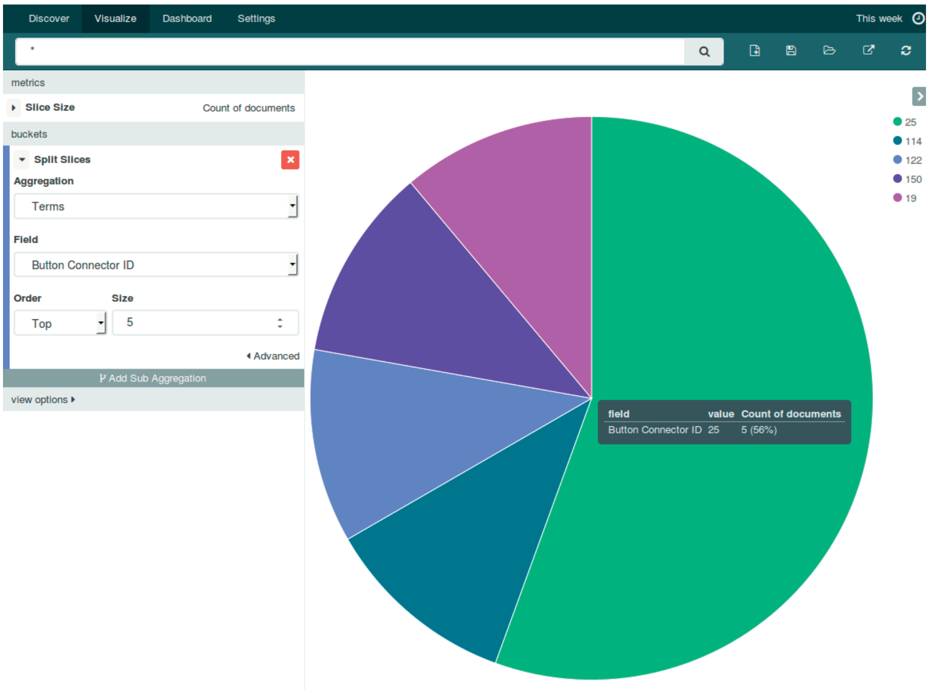
However, Elasticsearch is only storing the data and making it searchable. Therefore, Kibana was used as a dashboard for displaying the data from Elasticsearch. Through this dashboard, one can make queries and then visualize the results in various different forms. An example visualization is shown in Figure 11. In the visualization there is a pie chart illustrating how many times a specific button has been clicked.

6 Discussion

To discuss our findings, we next revisit our research questions one by one. In addition, we will also provide some directions for future research.

6.1 Research Questions Revisited

Based on our experiences with the proposed framework, we revisit the paper's questions as follows.



Slice Size: Count of documents

Aggregation: Terms

Field: Button Connector ID

Order: Top

Size: 5

field	value	Count of documents
Button Connector ID	25	5 (56%)

Fig. 11. Screenshot of a Kibana visualization.

RQ1: To what extent can a data collecting feature be implemented without compromising the evolution of the target program? Aspect-oriented approach to inserting additional features is quite unobtrusive by nature, which is supported by the code snippets. Also in this case, the usage monitoring facilities were inserted without changing the source code of the target application. The only parts which needed some modifications were the dependency addition to a build file and an insertion of an AspectJ file.

As these modifications were not altering the source code itself, the target application's evolution was not compromised nearly as much as with the manual approach. In this sense, if the target application's next version was to include new buttons, the aspect-oriented monitoring would notice them just as they did with all the rest. Therefore, the approach allows the target application to scale in that way without any additional efforts needed to include to the additional buttons as new data collecting points.

However, if the target application was to be changed in the way its buttons are instantiated, the aspect-oriented monitoring needs to be changed correspondingly. Even in this kind of a case though, the modification to the monitoring pointcut would most likely have to be done only once.

RQ2: What types of data can be collected with the given approach? With an aspect-oriented monitoring approach, pointcuts could be made on a vast variety of different points in the execution flow. For instance, we could have associated the pointcuts with the initialization of objects of a particular class, as well as any other public method. The same goes for advices, which can contain almost arbitrary code that is needed for monitoring.

Additionally, aspect-oriented techniques support various different types of data that can be collected. Software operation knowledge in general includes information such as in-the-field performance, quality and usage of software, and knowledge of end-user experience, and end-user feedback, and to some extent this is necessarily platform-specific. In our case, the Vaadin framework provides an API to get such data directly from the platform. For instance, there are straightforward methods to get information on timestamps, URI fragments, button captions, and so on. With such information, it is possible to gain knowledge for example about the clickstream a user leaves behind, the average time they spent on a specific page, or what kind of errors are logged the most.

All in all, the aspect-oriented approach provides us with the same flexibility in gathering different types of data as the manual approach did. With such arbitrary data types, the problems of analytics are more about asking the right questions than getting enough data.

RQ3: How to connect the data collecting feature with an analysis framework? Although collecting data can be done in most cases in a various ways, further exploring and analyzing of data might turn out more difficult. The use of a standardized analysis framework might require the data to be in a specific format. In this regard, the data logging tool's ability to unify the data it collects becomes important. In this study, Fluentd was used for collecting data, and it also performed the unifying by turning the data into the JSON format. This again was a format that the data storing solution supported and the visualization tool had an access to. Thus, the data collection tool's unifying feature enabled us to form an end-to-end analytics framework starting from the usage monitoring and peaking in the visualizations.

In circumstances such as these, general collection frameworks can provide a way to standardize parts of the logging even if data inputs and outputs varied from time to time. This becomes especially important when the aim is to combine data from different kinds of sources such as access, error and application logs.

6.2 Future Work

The work reported in this paper is only the very beginning of research regarding using aspects as a tool for analyzing user interactions. As already pointed out, at present we have a mechanism for collecting the data, and next challenge is to figure out which part of the data is truly meaningful, and how should the gathered data be used. Some of the directions for future work are listed below.

Extending the measurement point set. In addition to collecting straightforward data on user actions, broadening the focus to cover attributes such as in-the-field performance or end-user feedback can turn out as helpful opportunities for various different fields. For instance, a short user experience survey could be injected as an aspect into a specific point of execution flow, an error log could be sent to developers when a system crashes, or a sorry-note could be shown to the user in case of system performing under a specified level. Being able to perform this in a non-intrusive fashion could improve user experience considerably, with no risk to the future evolution of the system.

Experimenting with real-life apps. Obviously, the feasibility of the above data collection approaches is domain dependent, and the type of the application as well as the setup created for testing has an impact on whether or not operations are offline or real time. Therefore, experimenting the different approaches with real-life applications and developer needs forms an important part of future work. Our present strategy is to execute these experiments together with Vaadin and the associated developer community. In addition, once we reach a maturity level where the analysis framework can be used in production use, we wish to study how interaction data that has been automatically collected relates to user studies executed in more conventional fashion.

7 Conclusions

Fueled by the opportunities provided by the web and associated tools, analytics regarding the use of software applications have become a central aspect in software development. The rationale is that data regarding the fashion a software system is used helps in understanding the true needs of end users. This in turn enables the design of more satisfying software applications, with improved performance, simplified interactions, and superior user experience. However, gathering data on real-life use of applications is sometimes difficult, in particular when considering installable applications that cannot be easily updated remotely. Moreover, creating practical tools for analysis commonly requires application specific attention.

In this paper, we are experimenting how analytics facilities similar to web applications can be introduced to desktop and Rich Internet Applications written in Java. To keep the application intact from analytics facilities, we are using AspectJ as the implementation technique for introducing application-level monitoring, which allows us to hook analytics facilities to user interface events in a non-intrusive fashion. As for analysis, we are using an already existing tool set, where open source systems play a key role. The implementation we have created is concise, and it can be easily generalized to other applications if needed.

Based on our experiences reported in this paper, we find aspects a technique that is well-suited for creating data extraction features for already existing applications. In particular, given that the applications follow certain conventions, it appears to be relatively straightforward to create join points that are easily repeatable. Since we wish to track user actions, starting with user interface widgets is the natural starting point and almost all user interaction mechanisms in modern programs follow certain patterns, we believe that the results we have obtained can be generalized to many other environments, too. Moreover, already existing analysis tools provide support for filtering, analysing and visualizing data at real-time.

Acknowledgment

The authors wish to thank Digile Need4Speed program (http://www.n4s.fi/) for its support for this research.

References

1. Begel, A. and Zimmermann, T. Analyze This! 145 Questions for Data Scientists in Software Engineering. In Proceedings of the 36th International Conference on Software Engineering, pp. 12-23, ACM, 2014.
2. Buse, R. P., and Zimmermann, T. Information needs for software development analytics. In Proceedings of the 34th International Conference on Software Engineering, pp. 987-996, IEEE Press, 2012.
3. El-Ramly, M., and Stroulia, E. Mining software usage data. In Proceedings of 1st International Workshop on Mining Software Repositories (MSR'04), pp. 64-68, 2004.
4. Filman, Robert, Tzilla Elrad, and Siobhn Clarke. Aspect-oriented software development. Addison-Wesley Professional, 2004.
5. Grönroos, M. Book of Vaadin. Uniprint. 2011.
6. Highsmith, Jim. Agile software development ecosystems. Addison-Wesley Longman Publishing Co., Inc., 2002.
7. Hornbaek, K., Current practice in measuring usability: Challenges to usability studies and research. International Journal of Human-Computer Studies 64: 79-102, 2006.
8. Juergens, E., Feilkas, M., Herrmannsdoerfer, M., Deissenboeck, F., Vaas, R. and Prommer, K. Feature profiling for evolving systems. In Proceedings of the 19th International Conference on Program Comprehension, pp. 171–180, IEEE, 2011.
9. Kristjánsson, B., and van der Schuur, H. A Survey of Tools for Software Operation Knowledge Acquisition. Department of Information and Computing Sciences, Utrecht University, Tech. Rep. UU-CS-2009-028, 2009.
10. Matejka, J., Grossman, T., and Fitzmaurice, G. (2013, April). Patina: Dynamic heatmaps for visualizing application usage. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 3227-3236, ACM, 2013.
11. Metsä, J., Maoz, S., Katara, M. and Mikkonen, T. Using aspects for testing of embedded software: Experiences from two industrial case studies. Software Quality Journal 22(2): 185-213, 2014.
12. Pachidi, S., Spruit, M., van de Weerd, I. Understanding users behavior with software operation data mining. Computers in Human Behavior 30: 583-594, 2014.
13. Perry, Bruce W. Google Web Toolkit for Ajax. OReilly Short Cuts. OReilly, 2007.

PUBLICATION II

Collecting Usage Data for Software Development: Selection Framework for Technological Approaches

Suonsyrjä S., Systä K., Mikkonen T., Terho H.

In Proceedings of the Twenty-Eighth International Conference on Software Engineering and Knowledge Engineering (SEKE) 2016.

Collecting Usage Data for Software Development: Selection Framework for Technological Approaches

Sampo Suonsyrjä, Kari Systä, Tommi Mikkonen and Henri Terho
Tampere University of Technology, Korkeakoulunkatu 1, FI-33720 Tampere, Finland
{sampo.suonsyrja, kari.systa, tommi.mikkonen, henri.terho}@tut.fi

***Abstract*—Software development methods are shifting towards faster deployments and closer to the end users. Their ever tighter engagement of end-users also requires new technologies for gathering feedback from those users. At the same time, widespread Internet connectivity of different application environments is enabling the collection of this post-deployment data also from sources other than traditional web and mobile software. However, the sheer number of different alternatives of collecting technologies makes the selection a complicated process in itself. In this paper, we describe the process of data-driven software development and study the challenges organizations face when they want to start guiding their development towards it. From these challenges, we extract evaluation criteria for technological approaches to usage data collecting. We list such approaches and evaluate them using the extracted criteria. Using a design science approach, we refine the evaluation criteria to a selection framework that can help practitioners in finding a suitable technological approach for automated collecting of usage data.**

I. Introduction

One of the clear trends in the field of software development has been the ever tighter engagement of the end-users to the software development process. For example, methods such as Lean Startup [1] are dependent on more and more rapid feedback cycles. As described in a more general level in [2], the shift from Agile processes towards Continuous Deployment and experiment systems requires faster ways to validate the developed software than is possible with traditional communication methods, such as face to face conversations with end-users.

As these new methods are emerging, the whole software development process can be rearranged. In the aforementioned experiment systems for example, the deployment of software is not the end of the road for development efforts, but more of an initial step to start collecting data on user needs and then fine-tune the software [2]. With such approach, post-deployment data is first collected and then used for guiding the software development making the development process data-driven.

First and foremost this post-deployment data, such as data about how the system is used (i.e. usage data), has been used for guiding software development in environments like web and mobile development. In these contexts, constant connectivity – an important enabler for usage data collection – is the norm. However, breakthroughs of cloud software and Software-as-a-Service model, and the fact that most applications and platforms are Internet connected to begin with, are extending the use of data collection to a wider range of applications.

DOI reference number: 10.18293/SEKE2016-186

As this range of potential target applications, or programs whose usage data can be collected from, is getting wider, we are left with the challenge of finding the right technological approach for usage data collecting in the varying target application environments and cases. For example, manually adding code to target applications for logging purposes can be a straightforward option for developers in simple cases on one hand. But on the other hand, there are also different kinds of standardized tools and various approaches that among other things can automate this instrumentation or at least some parts of it.

To address this, we study what kind of challenges organizations face when they are starting the usage data collecting. Literature reviews along with a case study in an international telecommunication organization are used for finding these challenges, and they are extracted into evaluation criteria for data collecting technologies. We then describe several options for the automatic usage data collecting and evaluate them with the formed criteria. After this, we refine the criteria and the evaluated technological approaches into a selection framework that should help practitioners choose the suitable technologies.

The main research problem is *how to select the right technological approach for automated collecting of usage data?*. To address this, we derive two research questions from the main problem as follows.

- RQ1: How to evaluate different technological approaches for automated collecting of usage data?
- RQ2: What kind of technological approaches are there for the automated collecting of usage data?

The rest of the paper is structured as follows. In Section II, we take a look at the context of data-driven software development. Additionally, we go through the appropriate literature to find out challenges in automatic collecting of post-deployment data. Section III explains the formed evaluation criteria, and in Section IV we use the criteria to evaluate several technological approaches to usage data collecting. In Section V we derive a selection framework from the evaluation criteria and the technological approaches. In Section VI we draw some final conclusions.

II. Background

The background of this paper is two-fold. First, we address data-driven software development. Then, we introduce the challenges of automatic usage data collecting.

A. Data-Driven Software Development

As presented in [2], companies typically evolve their software development processes by climbing the *Stairway to Heaven* (StH). StH describes the shift from traditional waterfall development towards continuous deployment of software. The steps to be taken in the proposed chronological order are *Traditional Development, Agile R&D Organization, Continuous Integration, Continuous Deployment*, and the model ends up with *R&D as an Experiment System*. With each step, software development is becoming faster in the sense that it produces new releases of software ever more quickly. In the scope of this paper, the last phase is especially interesting as climbing the last step requires a fast-track of information from customers back to the development organization.

However, feedback gathering from customers is often slow, and sufficient mechanisms for it are missing. This can result in opinion-based development decisions. To ease the climb to the final step and make development more data-driven, Olsson & Bosch have developed the HYPEX model, i.e. *Hypothesis Experiment Data-Driven Development* [3]. In this model, *Minimal Viable Features* (MVF) are implemented and their expected behavior is described. A feature is implemented over many iterations and so the first 10% to 20% of its functionality is called an MVF. The deployed MVF is always instrumented to collect data on its use by customers, and this data is then compared with the initial descriptions of how the development organization thought that it would be used. Based on this *Gap Analysis*, the developers then either finalize or abandon the feature, or iterate the experimentation over again with a different hypothesis.

Such process has a lot in common with the *Build-Measure-Learn loop* (BML-loop) described in [1].Compared to the HYPEX model, the BML loop has many similarities and main differences are in the abstraction level. The BML loop is meant to validate the business feasibility of the product through the use of *Minimum Viable Products* (MVP). During each turn of the BML-loop a product hypothesis is formed and measurable metrics are linked to the hypothesis. The MVP is then built and the metrics are measured. Based on the outcomes of this data, the decision is made if the product development should be continued or another product hypothesis should be tested based on the experiences learned from the MVP.

As described above, both the HYPEX model and the BML-loop are data-driven approaches to software development. *Software Analytics*, as laid out in [4], highlights this use of data as well. However, this paradigm of analytics points out specifically the different types of analyses that are needed for turning the measured data into insights and eventually into development decisions. These are depicted in Figure 1. The model originates from the field of web analytics, but as its generality seems broad and with its experimental approach it should suit organizations well as a guidance in the *R&D as an Experiment System* phase of StH.

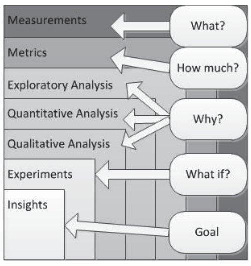


Fig. 1. Paradigm of Analytics (adapted from [4]).

All the aforementioned models include the phases of planning the data collecting, collecting the data, and analyzing the results to make decisions and iterating the process over again. In this sense, data-driven software development can be seen as an overarching term that typically consists of similar phases. To get a concrete definition from a technology standpoint and in the scope of this paper, we have formulated *Data-Driven Software Development* as an iterative process as follows.

1) *Planning of the data collection.* The goals of the analysis need to be known and the monitored applications and features should be selected based on them. The required resources, customer and user permissions and legal aspects of data collection need to be checked as well.
2) *Deployment of data collection.* The infrastructure of technical means to track the applications and collect post-deployment data needs to be installed.
3) *Monitoring of the applications.* The technical means can be internal to the application but also external - depending on the used run-time and platform technologies.
4) *Picking up the relevant data.* Monitoring should be configured to pick the data that is seen useful for the planned data collection and analysis.
5) *Pre-processing – filtering and formatting – the data.* The collected data is typically transferred to a remote location, but is typically filtered and formatted before sending to save resources.
6) *Sending and/or saving the data*. For effective analysis the data needs to be collected from long enough period and it needs to be available for the people working on the analysis. Often this means that hosting of the data storage is different from the applications. Thus, the system should transfer the data to storage either by means of continuous streaming or by saving it first to local cache and sending bigger amounts of data at the same time.
7) *Cleaning and unification of the data.* This process completes the work done by pre-processing described earlier but is necessary especially if data is flowing from various different sources.
8) *Storing the data.* Typically some database is used for storing the data.
9) *Visualizations and analysis.* A tools set helps stakehold-

ers to ask "what" and "how much" questions and to make conclusions.

10) *Decision making.* The results should lead to actionable decision for example on: new software development, user training, or marketing actions.

In this process, data collecting consists of phases 2-6.

B. Challenges of Automatic Usage Data Collecting

Fabijan et al. have described the challenges and limitations of customer feedback and data collection techniques in their literature review of software R&D [5]. The scope of their literature review included also manual and qualitative techniques such as interviews and observations, but the sources and challenges concerning automatic usage data collecting from the software product itself were as listed below.

- *Incident reports*: Available only after an incident.
- *Beta testing*: Only partially developed interfaces and functionality.
- *Operational and event data*: Security issues when such data is transmitted, potentially high amounts of data.
- *A/B testing*: Potentially confusing for customers when exposed to different versions.

Similarly, Sauvola et al. [6] described feedback gathering and its challenges as a part of software development companies' R&D efforts. Although their multiple-case study involved also many more feedback types than the automatically collected usage data, their descriptions of the cases implied various related challenges. We understood these as follows.

- *Permission checks.* The authors point out that in some specific domains the automated data collection from end-users is highly regulated and thus not executed at all.
- *Various sources of feedback.* Consolidating the feedback coming from various customers was seen as a challenge, and its processing relied heavily on its user's competence.
- *Only incident reports available.* Feedback was only gathered for troubleshooting purposes and not e.g. for improving existing products.
- *Systematic implementations are missing.* Although some mechanisms are in place to collect feedback and even product data, their implementations lack the systematic approach.
- *Difficulties to store, analyze, and integrate.* Even if feedback was gathered in most of the case companies, they reported having issues in storing, analyzing and integrating it back to the developers' processes.
- *Data availability and transparency.* The information about the collected data types as well as who and for what was it used was difficult to spread around the case companies. A reason for this, for example, was that the different parts of the development organization can see the data collecting as a risk as its use for new product development can cannibalize the current product markets.
- *Channels are not working.* As no systematic and organization-wide ways of feedback collecting were present, the feedback gathered in one place was regarded useless although it could have been in high value in the next place.

III. Evaluation Criteria for Usage Data Collecting Approaches

The challenges of usage data collecting are now extracted into evaluation criteria, which are fine-tuned based on the discussions with the case company. The challenges and limitations of usage data collecting can be consolidated as follows:

- *The amount of use cases for the collected data.* The number of use cases can be either too low or too high. Although there could be various uses for the same collected data, it might be used blindly to serve only a single purpose. On the other hand, the whole data collecting can face the critical challenge of trying to serve so many purposes and people that in the end it performs sufficiently to none of them.
- *The timeliness of the collected data.* Depending on the intended use, the timeliness of the data can form limitations to the collecting approach as well as to the source of data. For example, incident reports can be available only after incidents happen, and thus the use of such source has its natural challenges.
- *Continuous confusion for the users.* As mentioned, one of the well-known practices in the field of web development is A/B testing. Its implementation needs carefully planning, though. The more continuous the collecting is, the higher the risk of continuously introducing partially developed interfaces and functionality to users, who can find this troubling after a while. In addition, the collecting can affect the performance of the system.
- *Laws, regulations, and permissions.* Especially when the same data collecting approach is to be used in various different domains and countries, the related laws and regulations are going to be different for each situation. These checks for the data collection's legality take different amounts of time in each case thus enabling the data collecting in different cases at different times.
- *Privacy and security.* In addition to the overall permission checks, the security and privacy issues need to be addressed sufficiently by the collecting approach.
- *Various sources for the data collecting.* A high amount of sources creates a twofold challenge. The unification of the different types of data has to happen in a phase of its own (cf. phase 7 in Section II-A), or then the analyzer (cf. phase 9) has to have the capabilities to present and process the different types of data.
- *Lack of a systematic approach to collecting.* If a systematic approach is missing for the collecting, it is obvious that each phase of the data-driven software development is going to present new difficulties and challenges (e.g. difficulties to store, analyze, integrate etc.). These might be different in each case and they depend on the involved persons and their capabilities.
- *Availability, transparency, and usability of the collected data.* Even if the organization had first decided on what

kind of things they want to use the collected data, there are challenges also in how to make the data available, attractive, and usable for the right people. Thus, both the channels for distributing but also the tools for example for visualizing it (i.e. make the data usable) have to be sufficient enough for the selected audience.

We also analyzed these challenges from the perspective of the case company and used them as the basis of designing the evaluation criteria of usage data collecting approaches. The organization listed the challenges they felt were related to their case after discussing the overall topic of usage data collecting with us. Although each of the challenges they listed was found already from the list above, their descriptions of the challenges bring understanding to a more concrete level, which we try to emphasize with the examples linked to each criterion.

- *Timeliness.* When can the data be available? Does it have a support for real-time?
- *Targets.* Who should benefit from the data? What is the intended use? Does the approach support many targets? Does it produce different types of data or only one? ”Do we want results for troubleshooting or for new product development?”
- *Effort level.* What kind of a work effort is needed from the developers to implement the approach. ”How does the selected technology affect the production code? What is the work effort needed for the implementation?”
- *Overhead.* What kind of drawbacks are acceptable? ”How does the collecting approach affect the implementation environment, e.g. downtime and performance?”
- *Sources.* What sources of data can be used? Does the approach support many source platforms? ”Different kinds of technological environments – Where to focus our implementation efforts?”
- *Configurability.* How configurable the technological approach is? Can the collecting be switched on and off easily? Can it change between different types of data to collect? ”Is the collecting easy to switch on and off? Is the approach producing data in the right level of details?”
- *Security.* Can the organization who developed the collecting technology be trusted with the collected data? Is the data automatically stored by the same organization?
- *Reuse.* How can the technology be reused? Is it always a one-time solution or can it be reused as it is straightforwardly with another target application?

IV. Technological Approaches for Usage Data Collection

Next, we will go through a few technological approaches for the automated collecting of usage data. The abstract viewpoint is selected to not get stuck with the specific tools that happen to be around in 2016. Rather, the goal is to gain deeper understanding in how such tools and possible approaches work and how that is reflected in selecting them.

A. Manual Implementation

In the manual implementation the developer adds extra statements to the relevant locations of the software. On one hand, this highlights the flexibility of the approach – it does not limit the *timeliness, targets, sources,* or *security* in any way. On the other hand, adoption to new targets and sources would require significant rework making the *reuse* practically impossible. However, if additional functionalities such as run-time flags are added to the statements, switching the collecting on and off becomes significantly easier. This increasing *configurability* correspondingly increases the already high level of work *effort* needed for the implementation though. As a benefit, the approach almost guides the developer to collect data only from the intended sources, minimizing the *overhead* to the performance and of irrelevant data.

To conclude, there are only few real limitations with this approach. The needed work effort is high though, so e.g. if the code base is vast the approach can be come inappropriate. Therefore, we conclude that the approach is best suited either for the first few try outs with data collecting or for cases where the target and the source are particularly well focused.

B. Automatic Instrumenting with a Separate Tool

There are multiple tools, e.g. GEMS [7], that can automatically instrument the code for various data logging, quality assurance and performance monitoring purposes. This approach frees the programmers from the manual work and reduces the probability for errors lowering the *effort* significantly. Similarly, the *reuse* possibilities of the data collection should be high with automated tools since they are developed to work with different target applications in the first place.

However, the automatic tools are typically focused on only one type of *source* or *target*. Therefore, the *overhead* is likely to grow rapidly as the source cannot be set as specifically as with the manual approach. There are exceptions to this as well, and for example the framework presented in [8] balances its monitoring coverage with overhead automatically. Although these problems in general can be reduced by using highly *configurable* instrumentation tools when available, these criteria, along with *security and timeliness*, are almost completely intertwined with the specific tool selected. This highlights the inflexibility of the technological approach. The ideal case for this approach could be one with high importance in low implementation effort, such as a case with a huge code base, and with targets that need monitoring from the whole target application or even from many similarly developed target applications.

C. Aspect-Oriented Approach

Aspect-oriented approach is something of a mixture from the automatic instrumentation and the manual implementation. The research presented in [9] and [10] use aspect-oriented programming as a tool for code instrumentation. Further on, the use of aspect-oriented programming for usage data collection has been proposed in [11]. Additionally, in [12] the separation of similar monitoring code from the actual system code is highlighted, which could perhaps respond to the challenge of various data sources.

One important benefit of aspect-orientation is its expressive power. While automatic instrumentation is typically triggered by entering (or leaving) a function, the aspects can include more complex conditions for executing the data collection code. Aspect-based instrumentation allows the instrumentation to be system and application specific, which focus the collecting better on the relevant *targets*. This should also lead to optimized balance between the additional *overhead* and quality of the data.

The expressive power of AOP makes the approach similar to the manual implementation in its flexibility to create solutions that can be optimized by their *timeliness, configurability, and security* to suit any situation. On the other hand, the work *effort* needed for the implementation is not as high since the instrumentation is automated. However, learning to use AOP surely takes its toll if the developer is not familiar with the paradigm otherwise.

From the perspective of *reuse*, the aspect-oriented approach has both its limitations as well as benefits. If the different target applications are developed in such a similar fashion that the targeted data collecting places use the same syntax, the reuse should be very straightforward. Obviously, this sets a strict limitation to the reuse. On a more general level, the approach is depended on an available AOP library for the specific target application's programming language. If the language changes between the *sources*, i.e. the target applications, the reuse becomes much more difficult. This approach suits particularly well cases which need the same kind of system wide monitoring as the tool instrumentation's ideal case, but which at the same time require more flexibility from the collecting. An available AOP library for the case's programming language is obviously a critical limitation.

D. Alternative Implementation of a UI Library

An alternative implementation of a user-interface (UI) library can be set to automatically collect usage data. Because the user interaction is usually implemented with standard UI libraries, their components can be altered so that they include the collection of usage data within them. Similarly to automatic instrumentation, this approach frees the developers from the repetitive implementation *efforts*. Correspondingly, the issues are similar as well – data is easily collected also from unnecessary *sources* causing extra performance *overhead* and difficulties to the analysis phase. Although usage data is mainly linked to the UI and the types of data a UI library includes, some *targets* need integration with data types that are beyond the reach from these altered UI libraries.

On a more positive note, the approach has no limitations to the *security* or *timeliness*, and the *configurability* can be increased much like in the manual approach. As a matter of fact, this can be even easier if a differently altered UI library is deployed according to each requirements of a new case. The *reuse* of the implementations with this approach can be extremely easy, but new versions of the standard UI libraries create great issues as well.

TABLE I
SUMMARY OF THE TECHNOLOGICAL APPROACH EVALUATIONS.

	Technologies				
Criteria	*Man.ins.*	*Tool ins.*	*AOP*	*UI*	*E.E.*
Timeliness	+	-	+	+	-
Targets	+	-	+	-	-
Effort	-	+	+	+	+
Overhead	+	-	+	-	-
Sources	+	-	-	-	-
Config.	+	-	+	+	-
Security	+	-	+	+	-
Reuse	-	+	-	-	+

E. Execution Environment

The data collection can also be done by the environment without any modification to the application. For languages like Java and JavaScript the virtual machine is an execution environment where method and function calls can be monitored by instrumenting critical places. One example of such systems is Patina [13] where the user input like cursor movements and key-presses are monitored. In these approaches, the implementation *effort* is often low, but the produced data requires a lot of post processing and there may significant performance penalties since it will reduce possibilities for advanced just-in-time compilation.

This approach often has a limited set *sources* and *targets*, but within that limited scope *reuse* is good. Similar to the automatic instrumentation tools, the *configurability, security*, and *timeliness* of the approach are intertwined heavily with the specific implementation, i.e. tool, that is implemented.

V. Selection Framework for Automated Usage Data Collecting Technologies

We have summarized the evaluations of the technological approaches into the basis of the selection framework, i.e. Table I, giving each approach either a plus if it has a positive impact or if it does not restrict the implementation. An approach is marked with a minus sign if it limits the selection or the use of a data collecting implementation according to each criteria.

The first thing to do when selecting a technological approach to usage data collecting, is to rapidly **explore the case** to get a grasp of the most critical limitations to the technological approaches. These include things such as the size of the code base, availability of automated tools and AOP libraries for the target application's language and platform, and access to the UI libraries and execution environments.

If any critical limitations are faced, the next step is to **reject the unsuitable approaches** accordingly. For example, if there are many security issues related to the data being collected or if data needs to be sent in real-time, the 3rd party tools used in approaches B and E might have critical limitations that cannot be avoided.

The following step is to **prioritize the evaluation criteria**. In addition to the explored case information, one should find

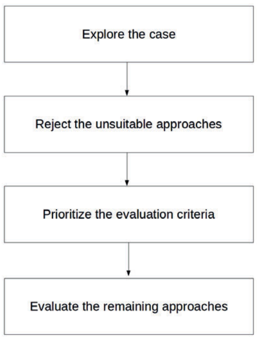

Explore the case
Reject the unsuitable approaches
Prioritize the evaluation criteria
Evaluate the remaining approaches

Fig. 2. Selection Framework for Technological Approaches.

out the goals different stakeholders have for the usage data collecting as these can have a major impact on the approach selection. If the goals are clearly stated, and the aim is e.g. to simply find out which of two buttons is used the most, manual instrumentation can work sufficiently. However, if the goal is stated anything like ”to get an overall view of how the system is used” or if the goal is not stated at all, the more automated and more configurable approaches most likely become more appealing. Therefore, one of the most crucial things to find out in this step is to understand what different stakeholders want to accomplish with the collected data.

After this, the final step is to **evaluate the remaining approaches**. The plus and minus signs used in Table I work as guidelines in this, but their emphasis obviously varies on a case to case basis. To summarize, the selection framework is illustrated in Figure 2.

The further evaluation of how the selection framework performs is clear choice for future work. The setting with the case company is interesting for them, but it is attractive also academically as it provides an environment to study the ”full-stack” that will be needed for the whole data-driven software development process in the end.

VI. Conclusions

In this paper, the main research problem was *how to select the right technological approach for automated collecting of usage data*. Literature reviews were performed to gain understanding of the context of the collecting processes and its challenges. These helped us form evaluation criteria for the technological approaches. We then described different approaches and evaluated them with the aforementioned criteria, which were then refined into the selection framework.

To summarize, the main contributions of this paper included literature reviews of the data-driven software development and of the challenges of collecting usage data, forming the evaluation criteria based on the studied challenges, description and evaluation of different technological approaches, and designing the selection framework for the technological approaches. The selecting of data collecting technologies is not a straightforward challenge but it needs to be addressed each time an organization wants to start the data-driven software development. Obviously, various contemporary tool evaluations are available both in academic literature and especially in practitioner publications, e.g. blogs. However, the abstract perspective of this study to the technological approaches rather than today’s tools should be valuable also in the long run.

Finally, the criteria described in this paper gives practitioners a good basis for the evaluation, but it works only as a guideline and case specific variations account heavily on the actual decisions. However, it provides them with a well-considered starting point in their journey towards ever more data-driven decision making.

Acknowledgment

This work is supported by Tekes (http://www.tekes.fi/) and Digile’s Need for Speed program (http://www.n4s.fi/en/).

References

[1] E. Ries, *The lean startup: How today’s entrepreneurs use continuous innovation to create radically successful businesses*. Crown Books, 2011.

[2] H. H. Olsson, H. Alahyari, and J. Bosch, “Climbing the” stairway to heaven”–a mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software,” in *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on*. IEEE, 2012, pp. 392–399.

[3] H. H. Olsson and J. Bosch, “From opinions to data-driven software r&d: A multi-case study on how to close the ’open loop’ problem,” in *Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on*. IEEE, 2014, pp. 9–16.

[4] R. P. Buse and T. Zimmermann, “Information needs for software development analytics,” in *Proceedings of the 34th international conference on software engineering*. IEEE Press, 2012, pp. 987–996.

[5] A. Fabijan, H. H. Olsson, and J. Bosch, “Customer feedback and data collection techniques in software r&d: a literature review,” in *Software Business*. Springer, 2015, pp. 139–153.

[6] T. Sauvola, L. E. Lwakatare, T. Karvonen, P. Kuvaja, H. H. Olsson, J. Bosch, and M. Oivo, “Towards customer-centric software development: A multiple-case study,” in *Software Engineering and Advanced Applications (SEAA), 2015 41st Euromicro Conference on*. IEEE, 2015, pp. 9–17.

[7] P. K. Chittimalli and V. Shah, “GEMS: A Generic Model Based Source Code Instrumentation Framework,” in *Proceedings of the Fifth IEEE International Conference on Software Testing, Verification and Validation*. IEEE Computer Society, 2012, pp. 909–914.

[8] J. Ehlers and W. Hasselbring, “A self-adaptive monitoring framework for component-based software systems,” in *Software Architecture*. Springer, 2011, pp. 278–286.

[9] W. Chen, A. Wassyng, and T. Maibaum, “Combining static and dynamic impact analysis for large-scale enterprise systems,” in *Product-Focused Software Process Improvement*. Springer, 2014, pp. 224–238.

[10] A. Chawla and A. Orso, “A generic instrumentation framework for collecting dynamic information,” *SIGSOFT Softw. Eng. Notes*, vol. 29, no. 5, pp. 1–4, Sep. 2004. [Online]. Available: http://doi.acm.org/10.1145/1022494.1022533

[11] S. Suonsyrjä and T. Mikkonen, “Designing an unobtrusive analytics framework for monitoring java applications,” in *Software Measurement*. Springer, 2015, pp. 160–175.

[12] M. Vierhauser, R. Rabiser, P. Grünbacher, K. Seyerlehner, S. Wallner, and H. Zeisel, “Reminds: A flexible runtime monitoring framework for systems of systems,” *Journal of Systems and Software*, vol. 112, pp. 123–136, 2016.

[13] J. Matejka, T. Grossman, and G. Fitzmaurice, “Patina: Dynamic heatmaps for visualizing application usage,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI ’13. New York, NY, USA: ACM, 2013, pp. 3227–3236. [Online]. Available: http://doi.acm.org/10.1145/2470654.2466442

PUBLICATION
III

Eeny, Meeny, Miny, Mo... A Multiple Case Study on Selecting a Technique for User-Interaction Data Collecting

Suonsyrjä S.

International Conference on Agile Software Development (XP) 2017. pp. 52-67.

Eeny, Meeny, Miny, Mo...
A Multiple Case Study on Selecting a Technique for User-Interaction Data Collecting

Sampo Suonsyrjä

Tampere University of Technology
P.O.Box 553, FI-33101 Tampere, Finland
sampo.suonsyrja@tut.fi

Abstract. Today, software teams can deploy new software versions to users at an increasing speed – even continuously. Although this has enabled faster responding to changing customer needs than ever before, the speed of automated customer feedback gathering has not yet blossomed out at the same level. For these purposes, the automated collecting of quantitative data about how users interact with systems can provide software teams with an interesting alternative. When starting such a process, however, teams are faced immediately with difficult decision making: What kind of technique should be used for collecting user-interaction data? In this paper, we describe the reasons for choosing specific collecting techniques in three cases and refine a previously designed selection framework based on their data. The study is a part of on-going design science research and was conducted using case study methods. A few distinct criteria which practitioners valued the most arose from the results.

Key words: agile software development, user-interaction data, multiple case study, software data collecting

1 Introduction

In the last few years, the world has witnessed a tremendous progress in the ways software is developed with. On one hand, this has already benefited both customers and vendors by improving productivity, product quality, and customer satisfaction [1]. On the other hand, the acceleration of release velocity has been such a strong focus point, that the evolution of the means of understanding user wants and needs could not have kept up the pace. For example, Mäkinen et al. [2] describe that customer data analytics are still used sparingly. Similarly, research related to the techniques of automatic collecting of post-deployment data and its use to support decisions still seems to be in its infancy [3]. This feels partly unfortunate, because agile software development has always had the intention of faster responding to changing customer requirements – and to achieve this, both rapid releasing and rapid understanding of customers are needed.

Addressing this, one of the promising solutions is to track users in the user-interface level, then analyze that data to understand how they use the software, and finally make decisions based on the analysis [4]. To start such a process, the first thing to do is to select a collecting technique that is suitable for the case. There are many restrictions to this, however, and these make the selecting a rather problematic task. Therefore, guidelines for evaluating and selecting a suitable collecting technique are needed. In our previous work [5], we have designed such a selection framework, which should serve as a guideline and help practitioners in these tasks. The objective of this study is to evaluate and refine that selection framework.

In this paper, we describe the reasons for choosing specific collecting techniques in three different case contexts and evaluate and refine the previously presented selection framework based on their data. The study is a part of ongoing design science research in which we have already designed the selection framework. This part uses the case study method to evaluate and refine the previous design and explore its contexts. Specifically, we address the research question:

– **What reasons software teams have for selecting a specific technique for user-interaction data collecting?**

To answer this overarching research question, we have derived two sub-questions. Firstly, the process of choosing a collecting technology will be explained. Secondly, we try to find out if some of the criteria we presented in our previous work are more significant than others or if there are completely other and more relevant reasons for choosing the technologies. The sub-questions for the study are declared as follows:

1. **How were the collecting techniques selected in each case?**
2. **What kind of criteria for choosing a certain technique were the most significant in each case?**

The rest of the paper is structured as follows. In Section 2, we present the background of the study, namely the selection framework which consists of selection criteria and a process. In Section 3, we explain how and why we used case study methods and describe the cases involved. In Section 4, we describe the process and criteria for choosing a specific technique for user-interaction data collecting in each case. In Section 5, we discuss those results to evaluate and refine the selection framework and in Section 6 we present the final conclusions of the study.

2 Background

To the best of our knowledge, related work for selecting techniques for user-interaction data is very limited. For example, a recently published systematic mapping study by Rodriguez et al. [6] identified the analysis of why certain technologies for monitoring post-deployment user behavior are selected over other similar existing solutions as a concrete opportunity for future work. However as a background for this study, we revisit the basics of the previously designed selection framework for user-interaction data collecting techniques.

The selection framework forms the basis for this study, as our goal is to evaluate the framework and refine it where necessary. It consists of a set of selection criteria and a process for the selecting. In addition, we introduce different techniques for user-interaction data collecting. These techniques and their evaluations are presented in a more detailed manner in [5]. They are mentioned nonetheless here for an overlook to the different alternatives that software teams have when they start collecting user-interaction data and for demonstrating the criteria part of the selection framework.

2.1 Selection Framework for a Collecting Technique

Criteria The selection framework guides software teams to evaluate user-interaction data collecting techniques in terms of the technique's *timeliness, targets, effort level, overhead, sources, configurability, security, and reuse.* In the following list, each criterion is described by demonstrative questions which could be asked as a team evaluates collecting techniques.

– *Timeliness.* When can the data be available? Does it have a support for real-time?
– *Targets.* Who should benefit from the data? What is the intended use? Does it support many targets? Does it produce different types of data?
– *Effort level.* What kind of a work effort is needed from the developers to implement the technique?
– *Overhead.* How does it affect performance, e.g. system response time to user-interactions?
– *Sources.* Does it support many source platforms?
– *Configurability.* Can the collecting be switched on and off easily? Can it change between different types of data to collect?
– *Security.* Can the organization who developed the collecting technology be trusted with the collected data? Is the data automatically stored by the same organization?
– *Reuse.* Is the collecting always a one-time solution or can it be reused easily?

Process The first thing to do when selecting a technique for user-interaction data collecting, is to rapidly **explore the case** to get a grasp of the most critical technical limitations. These include things such as the size of the code base, availability of automated tools and AOP libraries for the target application's language and platform, and access to the UI libraries and execution environments.

If any critical limitations are faced, the next step is to **reject the unsuitable techniques** accordingly. For example, if there are many security issues related to the data being collected or if data needs to be sent in real-time, collecting techniques using 3rd party tools might have critical limitations that cannot be avoided resulting in the rejection of the technique.

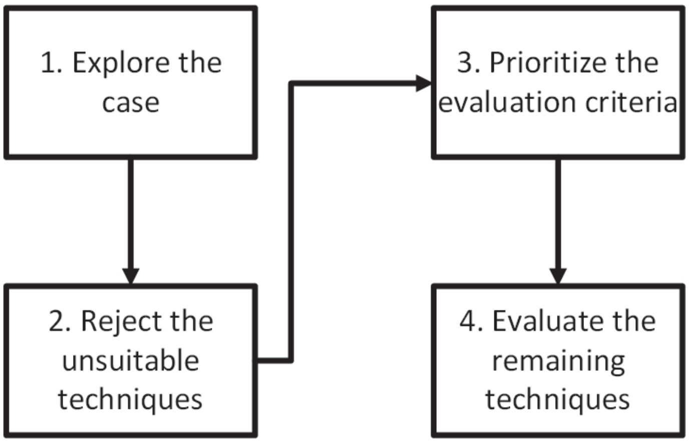

1. Explore the case
2. Reject the unsuitable techniques
3. Prioritize the evaluation criteria
4. Evaluate the remaining techniques

Fig. 1. Selection Framework for User-Interaction Data Collecting Techniques.

The following step is to **prioritize the evaluation criteria**. In addition to the explored case information, one should find out the goals different stakeholders have for the usage data collecting as these can have a major impact on the approach selection. If the goals are clearly stated, and the aim is e.g. to simply find out which of two buttons is used the most, manual instrumentation can work sufficiently. However, if the goal is stated anything like ”to get an overall view of how the system is used” or if the goal is not stated at all, the more automated and more configurable approaches most likely become more appealing. Therefore, one of the most crucial things to find out in this step is to understand what different stakeholders want to accomplish with the collected data.

After this, the final step is to **evaluate the remaining approaches**. The plus and minus signs used in Table 1 work as guidelines in this, but their emphasis obviously varies on a case to case basis. To summarize, the selection framework is illustrated in Figure 1.

2.2 Techniques for User-Interaction Data Collecting

Firstly, in **manual instrumentation** (Manual) developer adds extra statements to the relevant locations of the software. On one hand, this highlights the flexibility of the technique but on the other, adoption to new targets and sources would require significant rework making reuse practically impossible.

Secondly, there are multiple **tools for automated instrumentation** (Tools) of the code, e.g. GEMS [7], for various data logging, quality assurance and performance monitoring purposes. This technique frees the programmers from the manual work and reduces the probability for errors lowering the effort significantly.

Thirdly in between of the above techniques, **aspect-oriented programming approach** (AOP) is something of a mixture from the two. The research presented e.g. in [8] and [9] use aspect-oriented programming as a tool for code

Table 1. Summary of the technique evaluations.

	Techniques				
Criteria	*Manual*	*Tools*	*AOP*	*UI Lib.*	*E.E.*
Timeliness	+	-	+	+	-
Targets	+	-	+	-	-
Effort	-	+	+	+	+
Overhead	+	-	+	-	-
Sources	+	-	-	-	-
Configurability	+	-	+	+	-
Security	+	-	+	+	-
Reuse	-	+	-	-	+

+ = Supports selecting
- = Technique has limitations

instrumentation. Aspect-based instrumentation allows the instrumentation to be system and application specific, which focuses the collecting better on the relevant targets.

Fourthly, **an alternative implementation of a user-interface library** (UI Lib.) can be set to automatically collect user-interaction data. Because user-interaction is usually implemented with standard UI libraries, their components can be altered so that they include the collection of user-interaction data within them. Finally, the data collection can also be integrated into the environment without modifications to the original application. For languages like Java and JavaScript the virtual machine is an **execution environment** (E.E.) where method and function calls can be monitored by instrumenting critical places.

We have summarized the evaluations of different collecting techniques for the basis of the selection framework, i.e. Table 1, giving each technique either a plus if it has a positive impact or if it does not have restrictions in terms of the criterion. A technique is marked with a minus sign if it limits the selection or the use of a data collecting implementation according to a criterion.

3 Research Approach

The study was conducted using case study methodology. It allowed us to explore and describe the case specific situations and their circumstances related to the selection framework from deeper and more insightful viewpoints than if a research method with set variables had been used. Case study investigates contemporary phenomena in their real-life context [10], and this suited the purposes of the study well. The study is a part of on-going research effort, where we design, evaluate, and diffuse the selection framework by design science guidelines presented in [11] and with the process presented in [12]. The design science method of the underlying research effort affected this study as well especially in how actively the researchers had to take part in the cases. This participation was obviously required because the automated collecting of user-interaction data and its use for software development was still an unknown area for each of the case organizations. Moreover, the researchers had a substantial expertise considering the designed selection framework.

3.1 Explanatory Case Study

The selection framework, as presented in [5], includes predetermined criteria for evaluating the collecting technologies. These criteria could have been used straightforwardly as variables of a study with more experimental setting. However, the criteria have been derived from a literature survey and from only one case study. Therefore, we acknowledge that there can be other criteria that affect the selection as well, and perhaps with a greater impact. To allow the inclusion of these other possible factors into the selection framework, we have chosen to use specifically *multiple* case study method and gather data from three different cases.

This study uses *explanatory* case study methodology because its aim is at finding the reasons why software teams choose a specific collecting technology. The results of this explanatory case study are used for evaluating and refining the designed selection framework where necessary. Runeson & Höst [13] have categorized case studies by their purposes into exploratory, descriptive, explanatory, and improving. Since explanatory case studies are "...seeking an explanation of a situation or a problem, mostly but not necessary in the form of a causal relationship", their aims are well-suited for the study.

Case Selection Given the purpose of the studied selection framework, its potential users are mainly software teams that are only beginning to collect user-interaction data. This limited the potential cases for this study to software teams that had not yet selected a technique for user-interaction data collecting but were still willing to try such collecting out. Clearly, the selected cases had to be open enough that publishing the results reliably was possible and also accessible in the first place for the first author to do the research with them.

Similarly, the number of the cases selected for the study was affected by the fact that the first author had to spend considerable effort in each case. As suitable software teams for this study had not tried out user-interaction data collecting or explored its techniques, the first author had to have access to a potential software team to tell about the possibilities of such data collecting and initiate these tasks. All of these limited the number of selectable cases to few, and finally three software teams were selected for the study.

Data Collection The data was gathered from February to December 2016. Main parts of the data consist of meeting notes written down by the first author of this paper. Workshop type of meetings were held in each of the cases. Since collecting user-interaction data was a novelty for each participating software team, simple interviewing would not have worked. Rather, the meetings were organized as workshops where the first author motivated the software team to try out user-interaction data collecting and described the different possible techniques for doing so. In addition to the data gathered in meetings, the first author had designated work desks in the same rooms where the software teams were working in cases A and B. Therefore, data was also gathered by observation and by participating in informal meetings. However, these data were only used for verifying some of the previously collected meeting note data, such as how many standup meetings a team have in a week. Although these observational data were not collected in a formal fashion, for the first author it improves the reliability of the results in terms of data triangulation.

Validity and Reliability Considerations Although this study tries to investigate what kind of things have an effect on the decisions of software teams, the aim is not to find definitive proofs or certain amounts of statistical significance in these relations – rather to broaden the scope of possible causes. Therefore, the internal validity needs especially careful considering. Firstly, selections in earlier cases can have had effects on later ones. This was obviously not intentional but still surely possible because the same researcher explained the different options for the teams in each case. However, the author of this paper separated himself from the decision making in each of the cases and the decisions were made only by the software teams.

Secondly, the criteria presented with the selection framework can have guided the author of this paper to identify only those as the reasons for selection. Consequently, there can have been reasons that have not been mentioned aloud in the meetings but which still have had an effect on the decision. For example, a technology might have been seen as an unsuitable option in such an indisputable manner that the software team has not even mentioned it. This risk was mitigated in cases A and B by not only gathering data from meetings, but also by observing the working of the teams in the their offices and participating in their informal meetings.

The results of this study will not be generalizable for any software team. However, they provide a detailed look on the reasons these three software teams had for choosing a user-interaction data collecting technique. The three case organizations are different from each other in many ways, and therefore the results can give interesting insights to a wide audience. Although only one researcher gathered the data in each case, the meeting notes were shown to and accepted by team members in each case.

3.2 Case Organizations

Case A Organization A is a large international telecommunications company. The software team that was involved in this case consisted of around eight members. The border of one team in this organization is quite flexible as employees work for many products. The team members had titles of software architect, UX designer, software developer, and line manager. Their products consist primarily of software in the field of network management, and these range from Java software to web based systems. The software development method used in their team has some properties from agile development methods such as Scrum. They, for example, have bi-daily standup meetings and they use Kanban boards to organize their work. New versions of their product are released usually a few times a year.

Case B Organization B is a privately held software company in Finland. At the time of the study, they had around 300 employees and offices in three major cities in Finland and they primarily develop software in projects for their customers as ordered. The software team involved in this case, however, develops their own software-as-a-service solution. As in case A, the software team in case B also uses things such as daily standup meetings, Kanban boards and retrospective sessions familiar from some of the agile development practices. On the contrary however, they are releasing new versions of their product to the end-users far more often – usually biweekly. Their software team consists of seven members with titles such as product owner, UX specialist, software architect, and software developer.

Case C Organization C is a research and education center of around 10000 students and 2000 employees. The case C software team is part of a research group who have specialized in embedded systems design. They have developed Kactus2, which is an "open source IP-XACT-based tool for ASIC, FPGA and embedded systems design"[1]. The software has created traction from users world wide. It has been downloaded around 5500 times during the last year requests coming mainly from the USA and from middle Europe. The development team consists of four employees with the titles of software developer, software architect and business architect. The developed tool itself is an installable software system and installer packages for Windows and Linux tar-packages of its new versions are released three to four times a year.

4 Results

The results of the study are twofold. Firstly, we describe the processes with which the techniques were selected in each case. Secondly, we dive into the reasons the software teams had for their selection.

4.1 The Processes of Choosing a Collecting Technology

Case A In February 2016, members of the software team of Case A explained to the researcher that they had an overall interest in trying out the use of user-interaction data for the further development of their software products. The researcher had presented the different technological approaches for collecting such data in a previous informal meeting. These were the same approaches as described in [5]. Two of the software team's products had been then analyzed by

[1] http://funbase.cs.tut.fi/

the Organization A in terms of the suitability of the products in experimenting with user-interaction data collecting. The first of the two was Tool X written in Java, and the second one a JavaScript based Web-system Y. The team decided to carry on the collecting efforts with the System Y.

After this decision, the team had a meeting with the researcher to give a short presentation about the code base of the System Y and its software architecture. The meeting was arranged as a workshop to find out what kind of user-interaction data the team wanted to have collected. In addition, the team described what is important for the collecting technology and its implementation.

From this point on, the job of the researcher in the eyes of the software team was to develop a demonstrative collecting tool for their product. The researcher then used the criteria from the selection framework and was left with only one suitable technology approach – developing a new tool for **monitoring the execution environment**. After developing a prototype of such a collecting tool, the researcher presented it in a demo show for the team in March and got a thumbs up from the team to go on with experimenting with the actual System Y. A testing day with eight users from within the Organization A was held in December 2016 to try out an improved version of the collecting tool implemented in a lab version of the System Y. The developed collecting tool is available in GitHub[2].

Case B In case B, a similar workshop meeting as in Case A was held by the researcher with the software team in March 2016. The team explained the method they use for developing their software and what kind of a software the product is architecturally. It turned out, however, that this team had more experiences with collecting use related data even at that point. For example, they had tried out Google Analytics with some default settings for their product already. After explaining that the data was mainly collected for debugging, two of the team members and the researcher worked out also new targets in their software development process which could be improved with user-interaction data. These ideas ranged from prioritizing their product backlog to improvements in the user interface of the product.

The team was well motivated to try out user-interaction data collecting. However, as its return on investment was still unclear the first few tasks for data collecting were agreed upon to be completed with as little work effort and changes to the software architecture as possible. Therefore, three very specifically described places in the UI of the product were selected to be improved with the help of user-interaction data collecting. As the team had already tried out Google Analytics on the same product, it was a straightforward choice for the storing and analyzing the data of the tasks at hand as well.

At that point, the researcher described the same technological approaches to the team as in Case A. Also similar to Case A, the selecting of the collecting technology was an obvious pick since the three tasks were specified so explicitly. The team members and the researcher made an unanimous decision to use **man-**

[2] https://github.com/ssuonsyrja/Usage-Data-Collector

ual implementation for instrumenting the required places of the source code. The researcher was then given rights to change the source code. After applying the collecting code to six places in it, the version was sent to end-users for a two week collecting period in April 2016.

Case C In case C, an initial meeting was held with two members of the software team and the researcher in September 2016. Similar to the previous cases, the team members described the environment for which they develop software and the architecture of their product. The meeting then continued as a workshop, where each participant tried to figure out ways for how user-interaction data collecting could be used for their software development. Such targets were plenty, and no specific tasks were selected at that point. The researcher then explained the same technological approaches for user-interaction data collecting to the team members. The option of monitoring execution environment was rejected at this point, but the rest still remained possible for selecting.

The evaluation criteria from the selection framework were then used for the analysis of the product and its environment. Since the aspect-oriented approach raised the most interest among the software team, it was decided that the availability of AOP libraries and their suitability to the product were to be examined. An alternative implementation of a UI library was considered as a second choice, but the rest of the alternatives were rejected at this point. During the fall of 2016, the **aspect-oriented approach** was implemented technically successfully to the product. The first data collecting period is planned to be held during the spring of 2017 with a student group as experimental end-users.

4.2 Reasons for Choosing a Collecting Technique

Case A The first decision made by the Organization A was that they selected to try out user-interaction data collecting with System Y. This decision was based on **the sources and the reuse possibilities** of the collecting effort, because the motivation was to specifically try out this kind of data collecting as a technical concept rather than immediately produce actionable insights from exact places of a product. Had the collecting effort been carried out with the Tool X, the reuse would have been practically impossible since its environment was not as common as with the System Y.

Although the overall motivation was to test user-interaction data collecting conceptually, the team wanted to focus the requirements of the data collecting after the selection of the specific **source**, i.e. product. Finding a technology that could be easily reused with as little implementation **effort** as possible became a goal. This made the option of manual instrumentation heavily unfavorable. The team also emphasized how **the security and configurability** were important for the collecting technology. For example, the environment of their product was such that the collecting should be easy to be left out of the whole product when necessary. Consequently, the unobtrusiveness of the technology was highly valued.

Although the need for low configuring effort increased the attractiveness of using an automated tool for instrumentation, the security concerns were so heavy that the use of a tool developed outside the organization was not recommended. Therefore, the option of finding and using 3rd party tools was quickly rejected. In addition, the availability and effects of AOP libraries to things such as the **overhead** were unknown in the environment of System Y. Possibly the most significant of all, there was no motivation to make as big **a change to the software architecture** as needed by the aspect-oriented approach. The same reason applied for rejecting the option of an alternative UI library, because having different versions of the libraries was not acceptable for the delivery pipeline.

Case B Similar to case A, the motivation for the team of case B in user-interaction data collecting was to try it out as a concept. On the contrary however, this resulted in this case in a faster and a narrower scoped experiment. In other words, the **targets** and the **source** of their data collecting were very clearly defined in the first place. At the same time, this resulted in the lack of significance of the implementation **effort** because it would be so low even with the manual approach. Similarly, **reuse** was not considered as a significant reason, since there were no guarantees that the data collecting mechanism would be ever reused. All this resulted in a very straightforward choice of the manual approach. It was by far the easiest approach to implement on a small scale and it allowed the team to try out if user-interaction data collecting in a fast and low-effort way.

Case C Being a new thing for the case C software team, the user-interaction data collecting was again designed as a demonstrative experiment similar to the case A. Likewise, the interests of the team in this case were technical in the sense that they firstly wanted to find out a suitable technique for user-interaction data collecting. In the best case scenario, this technique could be then used with their actual product and actual end-users after the initial experiment. Because there was no simple access to experiment the collecting with real users, in the manner of case B, and the **security** requirements were weighted a lot heavier, the technical design of the collecting was the primary focus. Although the possible user-interaction data types and collection places, i.e. **sources**, were plenty, they were to be considered only secondly after validating the technical setup for the collecting.

This affected the evaluation of the collecting techniques in terms of prioritizing the criteria from the selection framework. Not limiting the **sources and targets** became important, because the collecting technique would not be selected and designed for just a one time try out. Although not mentioned out loud by the team, this could hint towards them valuing the **reuse** possibilities. All of these resulted in the attractiveness of the techniques enabling lower work **effort** spend on each distinct collecting place. Further on, the whole collecting was required to be able to be switched off as easily as possible. In other words, the **configurability** of the collecting technique was valued high.

5 Discussion

In each case, the process of choosing a collecting technology for user-interaction data was more or less the same. Members of the software team and the researcher had a meeting, where the researcher described the different technologies overall. After finding out what was the underlying goal for the team in the user-interaction data collecting, the most important criteria for the selecting became quite clear for both the researcher and the team members.

Comparing those criteria with the ones in the selecting framework, it is safe to say that most of the evaluation criteria from the selection framework were used without the researcher pushing the team towards those specific points. However, **timeliness** was never mentioned by the teams, which could signal either its insignificance or that its need is self-evident. On the contrary, **overhead** rose up in each case as a conversation topic but similar to the timeliness it did not seem to have any effect on the selecting in any case.

For both of these, it is worth mentioning that none of the techniques had a known disadvantage nor a limitation in terms of these criteria (timeliness and overhead) that would have been significant enough to get the whole technique rejected. However, in the original selection framework they were marked with minus signs for the monitoring execution environment technique. Therefore, the summary table with the evaluation criteria from the original selection framework, i.e. Table 1, requires some refining.

Firstly, the evaluations should consist of a wider scale than a plain plus or a minus sign. In these cases, some of the criteria affected the selection clearly a lot more than others. For example, the timeliness and overhead criteria did not seem to have an effect on the selection but on the other hand, the effort level of the manual technique had it rejected. Therefore, we propose an additional exclamation mark to the evaluations in case the criterion is a possible ground for a rejection. We have gone through the rest of the summary evaluations and added an exclamation mark where necessary based on the cases.

Secondly, some of the evaluations are not clearly pluses nor minuses. Therefore, we have added an option of +/- marking for the evaluation, if the technique does not definitely support nor limit the selection in terms of the specific criterion. Adding this option has had effects especially on the evaluations of the techniques that are heavily intertwined with specific tools. For example, the minus signs in the execution environment column of timeliness and overhead rows can be then replaced with this option. We have reviewed the evaluations and changed the original signs into +/- markings where necessary.

Thirdly, the *effort* criterion should be divided into two and renamed to *scalability.* The intention of the criterion is to depict the work effort that is required from the software developers to implement collecting snippets to the different places of the source code. Finally, however, there was a clear need for an evaluation criterion of how great an effort is needed from the software developers to change the software architecture and/or environment of the moment to support the collecting technique. This criterion could be named as the *change* that is

Table 2. Refined summary of the technique evaluations.

	Techniques				
Criteria	*Manual*	*Tools*	*AOP*	*UI Lib.*	*E.E.*
Timeliness	+	+/-	+	+	+/-
Targets	+	-	+!	-	+/-
Scalability	-!	+	+!	+	+!
Overhead	+	-	+/-	-	-
Sources	+	-	-	-	-
Configurability	+	-	+	+	+/-!
Security	+	+/-!	+/-	+	+/-
Reuse	-	+	-	-	+!
Change	+!	+	-!	-!	+

+ = Supports selecting
- = Technique has limitations
+/- = No clear support nor limitations
! = A possible ground the rejection

required. With these refinements to the criteria and evaluations, the summary table of the evaluations is as listed in Table 2.

In addition to the changes in the evaluations, the original selection framework requires some refinements based on the cases as well. First of all, in these cases the underlying goal of the whole collecting effort was the most important driver in the selection process. In cases A and C the delivery pipelines did not allow fast and flexible releases of new software versions with user-interaction data collecting capabilities, and so the software teams decided to develop their environment so that the collecting would be possible in the future. This became their real target, where as the team in case B did not have to develop their environment. On the contrary, they had the luxury of aiming straightforwardly at just testing out the collecting and the resulting user-interaction data with a minimum effort.

Therefore, the first step of the selection framework, *exploring the case*, should be clarified and replaced by a step of *defining a main goal* for the collecting effort. Based on these cases, it would be easy to then *remove the irrelevant evaluation criteria* after defining such a goal. For example, in case B the *scalability* of the collecting technique was seen unnecessary after the collecting was designed to be implemented as a one time solution.

Exploring the case still included important things that should be part of the selecting framework. Thus, the next thing of the process should be to *find out the critical limitations.* The rest of the original selection framework worked out as it was in these cases, and so no other changes were required to the final refined version of the selection framework. This framework is illustrated in Fig 2.

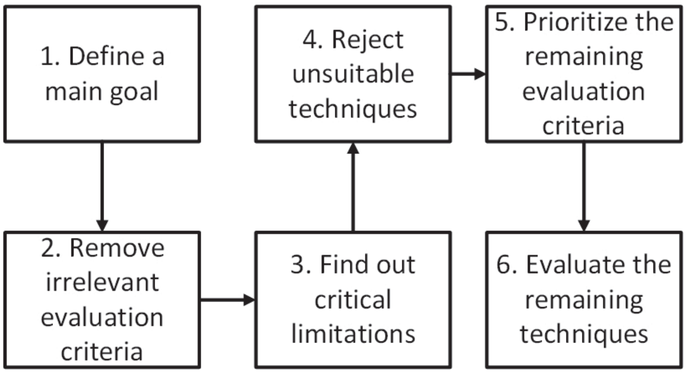

1. Define a main goal
2. Remove irrelevant evaluation criteria
3. Find out critical limitations
4. Reject unsuitable techniques
5. Prioritize the remaining evaluation criteria
6. Evaluate the remaining techniques

Fig. 2. Refined Selection Framework for User-Interaction Collecting Techniques.

6 Conclusions

In this paper, we studied three cases where software teams selected techniques for user-interaction data collecting. More specifically, we examined the reasons the software teams had for the selection. To complement this, we evaluated our previously designed selection framework and refined it based on the data gathered from the cases.

In these cases, two of the most valued criteria for the selection were the scalability of the technique and the lack of changes required to the software architecture and deployment pipeline of the moment. Additionally, teams appreciated the reuse, security, and configurability of the techniques as well as the support for a wide range of monitoring targets. On the other hand, the rest of the criteria presented with the original selection framework, i.e. timeliness, overhead, and support for different source applications, did not seem to have a significant effect on the selections.

The original evaluations of the different user-interaction data collecting techniques were refined to include markings for the different levels of significance. In addition, the original selection framework was fixed to better support these more detailed evaluations. With these changes, we think the selection framework and its complementary technique evaluations can help practitioners greatly to the beginning of their journey of user-interaction data collecting.

Acknowledgments

The authors wish to thank DIMECC's Need4Speed program (http://www.n4s.fi/) funded by the Finnish Funding Agency for Innovation Tekes (http://www.tekes.fi/en/tekes/) for its support for this research.

References

1. Leppänen, M., Mäkinen, S., Pagels, M., Eloranta, V.P., Itkonen, J., Mäntylä, M.V., Männistö, T.: The highways and country roads to continuous deployment. IEEE Software **32**(2) (Mar 2015) 64–72
2. Mäkinen, S., Leppänen, M., Kilamo, T., Mattila, A.L., Laukkanen, E., Pagels, M., Männistö, T.: Improving the delivery cycle: A multiple-case study of the toolchains in finnish software intensive enterprises. Information and Software Technology **80** (2016) 175 – 194
3. Fabijan, A., Olsson, H.H., Bosch, J.: Customer feedback and data collection techniques in software r&d: a literature review. In: International Conference of Software Business, Springer (2015) 139–153
4. Suonsyrjä, S., Mikkonen, T.: Designing an unobtrusive analytics framework for monitoring java applications. In: International Workshop on Software Measurement, Springer (2015) 160–175
5. Suonsyrjä, S., Systä, K., Mikkonen, T., Terho, H.: Collecting usage data for software development: Selection framework for technological approaches. In: Proceedings of The Twenty-Eighth International Conference on Software Engineering and Knowledge Engineering (SEKE 2016). (2016)
6. Rodriguez, P., Haghighatkhah, A., Lwakatare, L.E., Teppola, S., Suomalainen, T., Eskeli, J., Karvonen, T., Kuvaja, P., Verner, J.M., Oivo, M.: Continuous deployment of software intensive products and services: A systematic mapping study. Journal of Systems and Software **123** (2017) 263–291
7. Chittimalli, P.K., Shah, V.: GEMS: A Generic Model Based Source Code Instrumentation Framework. In: Proceedings of the Fifth IEEE International Conference on Software Testing, Verification and Validation, IEEE Computer Society (2012) 909–914
8. Chen, W., Wassyng, A., Maibaum, T.: Combining static and dynamic impact analysis for large-scale enterprise systems. In: Product-Focused Software Process Improvement. Springer (2014) 224–238
9. Chawla, A., Orso, A.: A generic instrumentation framework for collecting dynamic information. SIGSOFT Softw. Eng. Notes **29**(5) (September 2004) 1–4
10. Yin, R.K.: Case study research: Design and methods. Sage publications (2013)
11. Von Alan, R.H., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS quarterly **28**(1) (2004) 75–105
12. Peffers, K., Tuunanen, T., Rothenberger, M.A., Chatterjee, S.: A design science research methodology for information systems research. Journal of management information systems **24**(3) (2007) 45–77
13. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering **14**(2) (2008) 131

PUBLICATION
IV

Post-Deployment Data: A Recipe for Satisfying Knowledge Needs in Software Development?

Suonsyrjä S., Hokkanen L., Terho H., Systä K., Mikkonen T.

Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA) 2016. pp. 139-147.

Post-Deployment Data: A Recipe for Satisfying Knowledge Needs in Software Development?

Sampo Suonsyrjä, Laura Hokkanen, Henri Terho, Kari Systä, Tommi Mikkonen
Tampere University of Technology
P.O.Box 553, FI-33101 Tampere, Finland
Email: {sampo.suonsyrja, laura.hokkanen, henri.terho, kari.systa, tommi.mikkonen}@tut.fi

***Abstract*—In the field of improving software processes, one of the clear trends has been the ever tighter engagement of end users in the software development process. This is demonstrated by the shift from Agile processes to Continuous Deployment, which requires more rapid ways to validate the developed software and its value than is possible with traditional communication mechanisms and methods, such as face to face conversations with customers. While post-deployment data has been used for years as an extra data source – companies like Microsoft and Intuit have moved a few steps further from that already – we believe that there are numerous uncovered ways of taking advantage of post-deployment data in software development. In this paper, we study how automatically collected post-deployment data could be used for responding to knowledge needs of software development teams. The paper builds on data collected from a number of companies operating in Finland using a questionnaire study. The focus of questionnaire study was to approach post-deployment data – especially usage data – as means of getting information to support understanding of customer and end users.**

***Index Terms*—Software development, post-deployment data, agile software development, software usage data**

1. Introduction

One of the clear trends in the evolution of software development methods has been the ever tighter engagement of the end-user in the software process. As described in [1], the shift from Waterfall model and even Agile approaches to Continuous Deployment [2] requires more rapid ways to validate the developed software than is possible with traditional communication mechanisms and methods, such as face to face conversations with customers. At the same time, the recent advances in Internet connectivity of software products have been a key enabler in creating data-driven approaches to software engineering.

The goal of the development organizations is to streamline, optimize, and automate the development process to shorten delivery times and reduce costs. When operating in this fashion, considered sources of customer knowledge (e.g. feedback) may stabilize after a while, just like any repeatable process. This leads the software teams to use the same sources of knowledge over and over again, and possibly miss some crucial data or new sources of data, which become available only as products, markets, and development processes evolve.

To avoid getting caught standing still but rather take advantage of the increased availability of data, software teams have started to look for ways to utilize the data-filled environment the software teams work in anyways. For example, Begel and Zimmermann listed 145 questions software engineers would want data scientists to answer for them [3]. In their study, the two questions which were ranked as the most essential by the responding engineers, concerned the use of the products, or, in other words, the life cycle of the product after its deployment.

In this paper, we study *how software teams can utilize automatically collected post-deployment data in software engineering*, where term post-deployment data includes data concerning performance, quality, usage, and end-user feedback that are automatically collected after the commercial deployment of a software product. To reach this overarching goal, we review the appropriate literature and perform a questionnaire study which are then used for answering the three separate research questions we derive from the main question.

First of all, we study *what kind of post-deployment knowledge sources software teams use and which of these can be used automatically*. In other words, with this research question we try not only to find out what sources and data are used by such teams, but also to analyze how much of that could be collected automatically. To gather information for answering this research question we first investigate which knowledge sources are used in software development according to the scientific literature. After that, we continue with surveying the subject with the questionnaire.

Secondly, we find out *what kinds of utilization targets there are for automatically collected post-deployment data in software engineering*. To answer this, we review the literature for different types of post-deployment data and find out their use cases in different parts of software projects. In addition to the literature survey, we use the questionnaire to see what kind of new perspectives practitioners bring to the subject.

Finally, we study *what types of challenges there are in the utilization of automatically collected post-deployment* *data*. As there are most likely also challenges in the utilization of post-deployment data and its linking to the different parts of software projects, we again use our questionnaire survey and analyze its results to shed light on the topic.

The rest of the paper is structured as follows. In Section 2, we describe the background of this study by reviewing the literature on software development methods and the knowledge they require. In addition, we take a look at different types of post-deployment data. In Section 3, we describe the research approach. Section 4 explains the results of the questionnaire and in Section 5 we analyze how those results can be linked with the literature survey results. In Section 6, we make the final conclusion for the study.

2. Background

In this section, we first take a look at the evolution of software development methods over the last few decades. As general trends, these methods have formed the basis also for the processes of software teams. We analyze this progress in terms of how the needs and the sources of knowledge have changed. In addition, we assume that at least a few types of post-deployment data has been collected throughout the years alongside this aforementioned progress. In the latter subsection, we study what kind of data this has been, where it has been used, and what kind of challenges has been related to it.

2.1. From Pre to Post-Deployment Knowledge Needs in Software Development

In the most conventional software development methods, such as the Waterfall model [4] and the V-model [5], software development is considered to proceed in a linear way. The requirements gathering was done upfront in the beginning of the development, and software design was based on these documents. After this, software implementation and verification took place. Finally, when verification was done, the product was deployed, and the system ended up in the maintenance phase [6]. The initial requirements gathering phase depends on extensive documentation of the current customer needs. After these documents have been finalized and approved by the client, they are frozen, and the software is simply developed based on these requirements. The goal was to improve project management and control over the development organization [4]. The V-model extended this, linking post-deployment life stages to the initial steps where user requirements were collected. Still, in both these models, data is collected almost entirely before deployment, with only user acceptance testing done afterwards [5]. Furthermore, feedback from the use of the software is gathered only for maintenance purposes and bug fixing, not for addressing the core operation of the software – this was established beforehand in the requirements phase, and revisiting the decisions would mean a new software development project, starting with the new requirements. The techniques to collect these kinds of knowledge were traditionally outside of the software product itself. For example, Holzinger [7] has listed approaches to collect data regarding software usability. These include thinking aloud, field observation, and questionnaires.

Iterative and Agile methods were the next step in the evolution, where the relevance of changing customer demands was accepted and integrated into the development approach [8]. In iterative approaches, the first and most critical parts of the software can be taken into use before the remaining parts are implemented. There are several variants of the approach, where e.g. mainstream operations or risk management needs act as drivers for the development. In agile projects, customer requirements can evolve in parallel with the product [9]. Agile culture is steered by the guidelines in the agile manifesto [10], where communication with the client is given great importance. This communication provides information about the current state of the product to the customer, and customer provides the feedback and data on their opinions on its suitability. To support this, for instance when using Scrum [11], the project team develops small parts of the software and communicates this to the client to allow them to steer the development of the software asset. Moreover, it is possible to demonstrate how the design advances after each sprint.

Based on the above, agile software development teams and their customers might have the resources for continuous communication between the team and their customers and/or end users. However, once the project ends, so will the privilege of having the close communications come to an end. In addition, the close communication with the customers itself does not run without problems. For example, asking customers what they want can turn out to be a difficult task, as there is often a gap between what people say and what they do [12].

Continuous Deployment (CD) has changed the pace on which new software versions are released for use in the field [13]. Its aim is to get code deployed to the customer and end users as soon as it is developed. The benefits of CD are many, and include the reduced risk for each release and the prevention of producing wasted software [14]. However, if the developers rely only on close communication with their customer contacts during the project, then they may miss the feedback from the actual users of the system, in particular when the customers and end users are not the same. The *open loop* problem, presented by [15], describes in detail this asymmetrical situation of having means to continuously respond to changing customer requirements but not validating the software development efforts with continuous post-deployment knowledge from the customers and users. In such cases, the development teams have the technical capabilities to release new software versions extremely fast to respond to the new customer requirements they receive in meetings and discussions with the customers. However, the validation of the released software by the users or customers might in the slowest case have to wait for a qualitative analysis in the next face-to-face meeting, thus undermining the benefits of deploying continuously. Since tools play a key role in deployment, a lot of data is generated as a side-effect of the deployment in any case – Figure 1 presents a common setup – so introducing additional facilities for collecting usage data seems like the natural next step in the evolution.

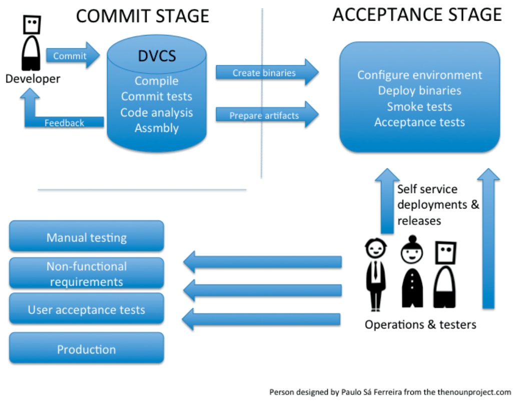


Figure 1. Anatomy of a Deployment Pipeline according to Jez Humble and David Farley [2]

To summarize, the needs of knowledge can be seen to have moved from pre-deployment phases of the software development processes towards post-deployment. At the same time, the knowledge gathering techniques used with especially these pre-deployment focused development methods are proving to be too slow on their own. The importance of for example qualitative user studies should not be rejected in any way, but there is a clear need to come up with new knowledge sources to accompany them.

2.2. Types, Uses, and Challenges for Post-Deployment Data

There has been a flock of *terms* defining the data that are gathered at the time of a software system is running. Already in 1981 Plattner and Nievergelt [16] published a survey in the field of *run-time software monitoring*. Later on, in 2010, van der Schuur et al. [17] defined *software operation knowledge* (SOK), which includes knowledge of in-the-field performance, quality, usage, and end-user feedback. More recently, in 2014, Olsson and Bosch have used the term *post-deployment data* for data that are generated by a product after its commercial deployment [18].

As a term, post-deployment data can be considered to highlight the link to the progress of deploying methods. This way, the contents of post-deployment data are aligned with what was defined with SOK, although SOK as knowledge is more like a result from data processing, including measurements, metrics, and analyses, not just raw data. Therefore, we have defined post-deployment data to include data concerning performance, quality, usage, and end-user feedback that are automatically collected regarding the product after its commercial deployment.

Collecting customer feedback has received a well-established position in research fields such as information systems, human-computer interaction, and participatory design [19]. For these fields, the utilization target has been on influencing customers and their behavior and on improving usefulness, ease of use, and user satisfaction. However, the domain of software engineering research has not given much attention to this topic that in any case seems important to anyone doing software development.

There are exceptions to this however. For example, using post-deployment data for fault-detection seems to have a long standing tradition (see e.g. [16], [20], [21]), and, as described above, run-time monitoring has been in use for over 40 years. In addition to software fault-detection, post-deployment has been utilized for goals such as profiling, performance analysis, software optimization and diagnosis, and recovery [20].

More recently, and especially within the domain of Software-as-a-Service, the developed software systems are increasingly connected to the Internet. Within these domains, post-deployment data and its uses such as A/B testing are recognized already as commonly deployed techniques [19]. In companies such as Microsoft and Intuit, these are used for continuous improvement of existing products, but also as an input to innovation and new product development [18].

Even with this well-established position of A/B testing, there are also challenges related to it and overall to the collecting of post-deployment data and its utilization. For example, [19] identifies such challenges in their systematic literature review of customer feedback and data collection techniques. Of those, the ones related to automatic data collecting and post-deployment data are listed in the following:

- Showing numerous versions to customers might get them hesitant as they are used to one version.
- Product expectations might not match with the experimental versions and thus the customer segments need to be chosen carefully to prevent revenue loss in case of operation problems.
- On-line ads and in-product surveys can be disturbing for users.
- Social networks can be used for similar purposes as product surveys, but they might generate vasts amounts of data making the analysis difficult. Moreover, the data from such sources can be biased for various reasons.
- Incident reports are available only after an incident has taken place.
- Operational and event data have been related to security issues at the time of transmitting the data.
- Large amounts of data can turn out to be a challenge to manage, analyze, and process.

3. Research Approach

To answer our research questions we conducted a questionnaire study within four Finnish software companies.

Questionnaire study was chosen as the data collection method to reach a sufficient number respondents that could take part in the study according to their own schedule but within a specific time frame. The focus of questionnaire study was to approach post-deployment data – especially usage data – as a means of getting information to support understanding of customer and end-users. Usage data was defined as ”automatically collected data from usage of a system”.

3.1. Questionnaire Study and Analysis

The questions were drafted by one researcher and then iterated with two other researchers to their final form. All the questions in the questionnaire were open ended which served our purpose of gaining responses that cover widely aspects of post-deployment data. In addition of the these questions, background information from each respondent was gathered. The five questions of the questionnaire are listed below (Q1-Q5):

- Q1: What are the most important sources of knowledge in understanding the needs and problems of your customers/end-users?
- Q2: What kind of additional knowledge would you like to have about your customers and end-users?
- Q3: What kind of benefits are you hoping for with usage data?
- Q4: How could you utilize usage data for your own work?
- Q5: What kind of challenges do you think there are in the utilization of usage data?

The collected raw data was processed for analysis. First, each response was coded with respondents' identification that indicated also the company where respondent worked at. Secondly, color-coding was applied to indicate the question that response was given to. Empty responses were removed from the data set. Together 210 responses were taken for analysis.

Data analysis was done in three phases. In all phases, analysis was done by utilizing iterative thematic coding. The three phases were established based on the interview questions; the first phase regarded responses to question Q1, second phase analysis combined responses of questions Q2, Q3 and Q4, and finally, the last phase of analysis was conducted for responses of Q5. Thematic coding was conducted by two researchers by first reading all the responses after which each response was individually discussed and assigned to a sub-theme. Sub-themes emerged from data either at time of reading all the responses or while handling individual answers. Sub-themes where then combined to create main themes of the analysis.

3.2. Study participants

Four software companies took part in the questionnaire resulting into 25 responses all together. The questionnaire was distributed to a company contact person that internally invited employees to take part. Anonymity of respondents was guaranteed, and responding was voluntary. Target audience was defined as software professionals that were designing or developing software products and had the possibility to influence plans and design of the product. Background information of respondents is presented in table 1. Age range of respondents was between 24 and 44 years.

TABLE 1. ROLES OF RESPONDENTS FROM DIFFERENT COMPANIES.

Company	Roles of respondents
A	software architect (2)
	software designer
	software developer
B	project manager
	creative director
	software developer
C	chief business officer
	software designer
	software developer
	UX designer
	service manager (2)
D	software designer (5)
	software developer (4)
	software consultant
	UX designer
	visual designer

Since our questionnaire study was conducted in Finland with a limited number of respondents from four small and medium sized software companies, the results cannot be generalized but rather require additional research from other countries and from different kinds of software teams. However we believe that presenting such data originating from the industry is needed in order to inspire new research to foster the possibilities of utilizing post-deployment data in software development.

4. Questionnaire Results

In this Section, we describe the results of our questionnaire to software project personnel.

4.1. Knowledge Sources in Software Projects

Gaining knowledge about customers and end-users involved direct contact with customer, information collection from users, and as a third option respondent's information gathering from other available sources.

Customer contact provided information by means of face-to-face meetings or discussions via different channels. Nine responses included customers as the source of information without specifying how information was passed from customer. While customer meetings was mentioned three times as the source of gaining knowledge, discussions was mentioned seven more times. For discussion, it was mentioned that an active product owner with continuous contact helps in understanding the customer. One respondent concluded that discussions with customer lead to understanding

TABLE 2. CATEGORIES AND THEIR RELATED THEMES THAT ROSE FROM ANSWERS TO QUESTIONS 2,3, AND 4

Data types	Analysis options	Utilization targets
Time	Value analysis	Development options
Performance	Users' problems	Actual needs
Amount of use	Use paths	Resourcing
Errors	Actual use	UX
	User profiling	UI design

of use cases that are relevant to customer. Customer contact was the most common source of information resulting from Q1 yet the channel remained ambiguous in many answers.

User-centered information was recognized as one of the main themes rising from the responses to Q1. Even though customer was mentioned as the source of information more often, seven responses fell under theme of user research. Means to collect information of users were interviews with six mentions, while also observation (2) and getting to know the context also came up. Analytics from end-users was mentioned by one respondent. Feedback was the second theme under user-centered information including both feedback about the product and in the form of support requests.

Own information gathering was divided into two themes: documentation and other information acquisition. For documentation, four respondents mentioned four different types of documents: issues documented, statements from business consults, existing documents from customers, and one mention of documents without specifying their type. Means for other information were general familiarization with the topic at hand, using Google, finding out about existing solutions, and as last, using the product that was under development.

4.2. Uses for Post-Deployment Data in Software Processes

Questionnaire answers to questions 2, 3, and 4 were combined for a bottom-up analysis. However, here we will go through the formed categories and their themes with a top-down approach for easier comprehension. Three top-tier categories were formed, and these could be looked at as three stages of knowledge, i.e. data, information, and knowledge. Each of these categories and their themes are listed in Table 2 and described in more detail in the following.

First, different types of post-deployment data formed one category from the answers. Answers mainly came from the third question. In this category, there were only 2 answers to Q2, the most significant question for this being Q3 with 28 answers. 10 answers were to Q4. The third question was about the benefits of post-deployment data and in that sense, the respondents seem to have valued even the different data types on their own without additional analyses. These data types that were formed as the initial themes were time, performance, errors, and the amount of use.

1) Answers in the **time** theme ranged from ponderings such as ”How long it takes for users to complete their goals” to straightforward questions like ”On what time of a day do our users use the system?” and ”Where on our system do the users spend their time?”. There were 11 answers in the time theme.
2) **Performance** related answers included data types such as load and stress measurements, performance issues, and response time. In addition to straightforward data types, the answers in this theme also included links to data utilization targets such as ”more efficient programs”, ”optimizing performance”, and ”faster and smarter functioning systems”. This theme was formed of 8 answers.
3) The theme of **amount of use** included 11 answers. Of these, 10 were to the Q3, which was related to the benefits of post-deployment data. 9 answers mentioned clearly the amount of use from e.g. features and functionalities, but 3 had included also further going analysis needs and targets such as ”Can users find new functionalities?” and ”optimizing the most used functionalities”.
4) **Error** related theme consisted of 10 answers. Systems' exception and error messages as well as their amounts were mentioned in the answers. Bug fixing was seen as the target for using these data in 4 answers.

Secondly, a category of different analyses rose from the answers. These were weightings of how beneficial e.g. features and functionalities are, what kind of problems users walk into, what kind of paths users take in the software, how users are actually using the software, and how users could be profiled. This category was more evenly balanced with 10 answers to Q2, 16 to Q3, and 11 to Q4.

1) The theme of **value analysis** with 10 answers consisted of analyzing both the most and the least valuable functionalities, features, and use cases. Although this theme had answers to each question (Q2-4), Q3 was clearly the most significant source with 8 answers.
2) **Users' problems** related answers formed a theme of 9 answers. One answer mentioned ”pain points”, which describes this theme quite well as each of the answers was in some way mentioning the need for finding or recognizing difficult points in systems.
3) Answers concerning **use paths** made up a theme of 7. Of these, 4 answers were to Q3 and 3 to Q4. This left this theme as the only one without answers to Q2 concerning the additional knowledge project teams would want to have. However, use path analysis was mentioned as a way to utilize post-deployment data in 2 respondents' work in order to ”identify critical paths” and ”speed up workflows”.
4) Contrary to the former, the answers in the **actual use** theme consisted only of answers to questions 2 and 3. Two of the 6 answers mentioned ”actual”

use of system and hence the name of this theme. All in all, these answers highlighted the need for access to or feedback from the real end-users.

5) The theme of **user profiling** was formed of only 4 answers. Each of these came as an answer to Q2. These additional knowledge needs included puzzles such as "What kind of users do we have?", "How do the young use services?", and "What channels are the users using?".

Thirdly, a category of targets on how post-deployment data could be used was formed. These different parts of software projects were development opportunities, eliciting actual needs from users, resourcing, user experience, and user-interface design. 19 of the answers were to Q2, 7 were responses to Q3, and 17 to Q4.

1) 9 answers formed a theme of **development options**. None of the answers came from Q2, and so with the 3 answers to Q3 and 6 to Q4 this theme represented the benefits of and use cases for post-deployment data. These were mostly targets such as optimizing and improving software systems, finding guidelines for new designs and points in need of redesign, and supporting decision making for both business and software development.
2) Contrary to the former, the theme of **actual needs** consisted of 11 answers to Q2 and 2 answers to Q3. The answers in this theme included straightforward knowledge needs such as "Which browsers do they use?" and "What kind of an action environment is it?". However, the answers point clearly towards a more significant information gap between the respondents and the end-users. For example, there were answers such as "..no connection to the end-users, and so it's difficult to understand how the system should actually function", "Understanding the the language of the customers, what they actually need", and "If a customer produces the service to end-users, the end-users' needs are often left in the background and customer designed use cases get highlighted."
3) **Resourcing** related answers formed a theme of 3 answers to Q2 and 6 to Q4. Prioritizing the tasks of the software team, focusing to the most important features, and cutting out the useless functionalities were mentioned in the answers.
4) There were 6 answers in the **UX** related theme: 4 from Q2, and 1 from both Q3 and Q4. The answers ranged from specific needs such as "to recognize points where users boil over", to more overall targets related to UX such as "..the automatically collected data should support us identify if a feature is made well and usable."
5) An answer to Q2 mentioned the need for additional knowledge to support making service easy enough to use. On the other hand, an answer to Q3 proposed usage data as an insight generator for finding out the "clearness of UI and functionalities". The theme of **UI design** consisted of 6 answers, of which the rest 4 came from Q4. These mentioned either straightforwardly "UI design" or more specifically "Highlighting the useful but rarely utilized features".

4.3. Challenges of Utilizing Post-Deployment Data

Themes in this section consisted of answers to question 5. All in all, 9 themes rose from these 46 answers, and these themes were again used for creating three top level categories. These categories were acquiring data, data processing, and maturity. In the following, we will describe each category and their themes in more detail.

Firstly, the category of challenges in *acquiring data* consisted of three themes: collecting, collecting techniques, and data quality and quantity. This category included 16 answers altogether.

1) The answers in the theme of **collecting**, plainly mentioned how the collecting can be a challenge. One of the answers described also how the access to data can turn out to be a problem. The total amount of answers in this theme was 5.
2) There were 4 answers in the theme of **collecting techniques**. The respondents found challenges such as "How to collect data from the products that are in the field?" and "Finding the right tools and their implementation".
3) The vast quantities of data was also seen as a challenge in four answers. Of these, three answers had mentions of how that might result in challenges for filtering and analysis. In addition, finding the right types of data and the quality of the data were mentioned in this theme of **data quality and quantity**. The total number of answers in this theme was 7.

Secondly, the category of *data processing* rose from the answers. This category included three themes, which were based on 17 answers. The themes were tools, analyzing, and utilization.

1) Four answers formed the theme of challenges related to **tools**. Both the collecting and the analysis parts were mentioned in the need of better and easier to use tools. In addition, the reusability of the collecting infrastructure was mentioned in one of the answers.
2) The answers in the theme of **analyzing** either mentioned plainly how analyzing can be a challenge or highlighted how data on its own is enough but analysis is required on top of that. This theme consisted of 6 answers.
3) 7 answers formed the theme of challenges related to **utilization** of usage data. The answers in this theme highlighted how there needs to be a utilization target for the collected data. Otherwise, "...the data will often be insufficient" and "finding time and money" can turn out to be a challenge.

Thirdly, the themes for challenges related to *immaturity* were formed from the answers. These themes were customer immaturity, process immaturity, and privacy. All in all, there were 13 answers in this category.

1) The theme of **customer immaturity** highlighted clearly the need for justifying the value of usage data to the customer. However, there were only 3 answers in this theme.
2) **Process immaturity** emerged as a theme with four answers. For example, answers mentioned things such as ”This is a fairly new approach and still not known in projects” and ”The costs of implementing usage data and its studying should be also budgeted in the projects”.
3) Finally, we included the challenges related to **privacy** to the maturity category as it can be seen to reflect the immaturity of the end-users and law-makers on the subject. This theme consisted of 6 answers.

5. Discussion

Based on the results of the study, we summarize the findings regarding the needs, current practices and attitudes towards the use of post-deployment data in software engineering. In the following, we revisit each research question to connect the present-day literature with the questionnaire survey results.

RQ1. What kind of post-deployment knowledge sources software teams use and which of these can be used automatically?

The questionnaire answers indicated how the discussions and meetings with customers were the main knowledge sources for the responding software development companies. This suggests that these companies rely on their teams (and customers!) to have the time to have these meetings and discussions, which again is probably happening only while a specific project between the two is on-going. As seen in the literature review, this kind of knowledge sources characterized mainly the development efforts with the Agile methods.

For every case, there is obviously no reason to completely remove the feedback gathering in discussions and meetings with customers, but its value surely can get stressed too much. For example, it is easy to see how it highlights only the opinions of the people present in the meeting. However, if the same feedback could be collected from a sufficient portion of the users, it would naturally represent them more evenly. To do this efficiently, automation is needed. Our literature survey mentioned questionnaires, and such could be seen as a technique to at least collect data automatically, even if the analysis needed more effort.

In the same manner, the technique of observing users in the field presents a twofold case for automatic use. Both our questionnaire and the literature survey described field observation as a knowledge source for software teams. In these cases, the technique was intended to rely on either physically going next to the users or record them on a video to observe how they use the product. This obviously requires a high amount of manual work, which makes it practically impossible to observe all the users. However, especially by being physically at the same place with a user, the inadvertent gathering of different kinds of additional qualitative knowledge can help software teams to understand their users and use environment. If, on the other hand, observing is concerned with only specific predetermined quantitative data, it could be collected automatically as well. The automatic collecting should also come with benefits such as collecting data from a wider audience, having less human errors e.g. typos, and collecting much more data in less time.

On one hand, the sources with an obvious need for automatic data collecting, i.e. analytics, were mentioned only once by the respondents. In this sense, they are not a highly valued knowledge source at least for the responding software teams. On the other hand, additional knowledge sources are clearly needed as can be seen in the *actual needs* theme in the answers. For example, the difference between customers and end-users was highlighted in some answers meaning that customers only represented themselves and there was no access to the actual end-users. Moreover, even if the feedback was received from the end-users, there can be gaps between what people say and what they do as described in the literature and also in the answers.

RQ2. What kinds of utilization targets there are for automatically collected post-deployment data in software engineering?

The first category that rose from the questionnaire results was the different types of data. It was formed of the themes of time, performance, amount of use, and error. With these, it can be seen to include much of the same types of data as the definition we formed for post-deployment data with the help from the literature.

Most of the different types of analyses that formed the second category have been seen also in the literature. User profiling, use path analysis, and users’ problems detection were the clear examples of these. The themes of actual use and value analysis were a bit more abstract, although for value analysis some of the answers suggested a straight-forward conclusion from high use of features to their high value. Obviously, the utilization of the data from the use of features was highlighted in this theme of value analysis.

The category of the targets on how post-deployment data could be used was in line with the literature, but offered some new ideas as well. The targets in the themes of UX, UI design, and development options have been discussed in their respective fields of human-computer interaction and software product development. Likewise, the theme of the actual needs highlighted the difference between customers and end-users, which is a well-known problem as well.

However, this is a critical problem to which the utilization of automatically collected post-deployment data could bring some help. Additionally, the theme of resourcing raised target ideas such as prioritizing the development of features and fixing of bugs.

RQ3. What types of challenges there are in the utilization of automatically collected post-deployment data?

The first challenge category in the questionnaire answers was related to data acquiring. The problems with vast quantities of data are well-known challenges also seen in the literature, e.g. concerning big data, and they are mentioned in this context as well [19]. Similarly, the challenges of collecting and analyzing data have mentions both in our questionnaire answers as well as in the literature.

But even if the challenges of reusable and easy-to-use infrastructure for collecting and analyzing were resolved, a more significant problems need to be solved in the first place. One of the challenges mentioned in our questionnaire answers was to not have a utilization target for the collected post-deployment data. For methods such as the Lean Startup, the first step is to form a hypothesis that is then tried to validate. However, if data is collected without the purpose to validate such a proposition, the data collection is most likely done in vain. Additionally, this might prove to be a dangerous point of setting vanity metrics [22].

The category of challenges related to immaturity is especially interesting with the business model of our questionnaire answering software teams, who develop software as outsourced projects. As mentioned, the steps towards the utilization of post-deployment data only have to be taken by one organization in the case of product businesses. But for outsourced software development projects, the same utilization requires efforts and changes also from their customers.

6. Conclusions

The interest in such post-deployment data has been around for more than 40 years. Data from operations such as run-time monitoring have been used for various purposes, including in particular debugging and problem analysis [20]. However, the recent steps forward by companies such as Facebook [23] or Intuit [24] indicate that there are still many new ways of taking advantage of different kinds of post-deployment data.

In this paper, we propose including the concept of post-deployment data utilization in the context of knowledge needs in software engineering. The main research question was: how software teams can utilize automatically collected post-deployment data in software engineering? The question was approached with literature reviews and a questionnaire survey.

The main finding was that there are many possible utilization targets for post-deployment data in software projects. There are places where automatically collected data can streamline the communication between software teams and end-users but more importantly, the teams can understand a wider audience than before. Although more specific research on each utilization target is needed, we think that post-deployment data can provide new intriguing knowledge sources, which are clearly needed in software engineering as development methodologies move more and more towards the end-users.

Acknowledgments

The authors wish to thank Digile's Need4Speed program (http://www.n4s.fi/) funded by the Finnish Funding Agency for Innovation Tekes (http://www.tekes.fi/en/tekes/) for its support for this research.

References

[1] H. H. Olsson, H. Alahyari, and J. Bosch, "Climbing the" stairway to heaven"–a mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software," in *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on*. IEEE, 2012, pp. 392–399.

[2] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader)*. Pearson Education, 2010.

[3] A. Begel and T. Zimmermann, "Analyze this! 145 questions for data scientists in software engineering," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 12–23.

[4] W. W. Royce, "Managing the development of large software systems," in *proceedings of IEEE WESCON*, vol. 26, no. 8. Los Angeles, 1970, pp. 1–9.

[5] K. Forsberg and H. Mooz, "The relationship of system engineering to the project cycle," in *INCOSE International Symposium*, vol. 1, no. 1. Wiley Online Library, 1991, pp. 57–65.

[6] H. D. Benington, "Production of large computer programs," *IEEE Annals of the History of Computing*, no. 4, pp. 350–361, 1983.

[7] A. Holzinger, "Usability engineering methods for software developers," *Communications of the ACM*, vol. 48, no. 1, pp. 71–74, 2005.

[8] C. Larman and V. R. Basili, "Iterative and incremental development: A brief history," *Computer*, no. 6, pp. 47–56, 2003.

[9] B. Boehm, "Get ready for agile methods, with care," *Computer*, vol. 35, no. 1, pp. 64–69, 2002.

[10] M. Fowler and J. Highsmith, "The agile manifesto," *Software Development*, vol. 9, no. 8, pp. 28–35, 2001.

[11] K. Schwaber, "Scrum development process," in *Business Object Design and Implementation*. Springer, 1997, pp. 117–134.

[12] E. Backlund, M. Bolle, M. Tichy, H. H. Olsson, and J. Bosch, "Automated user interaction analysis for workflow-based web portals," in *Software Business. Towards Continuous Value Delivery*. Springer, 2014, pp. 148–162.

[13] M. Leppanen, S. Makinen, M. Pagels, V.-P. Eloranta, J. Itkonen, M. V. Mantyla, and T. Mannisto, "The highways and country roads to continuous deployment," *Software, IEEE*, vol. 32, no. 2, pp. 64–72, 2015.

[14] G. G. Claps, R. B. Svensson, and A. Aurum, "On the journey to continuous deployment: Technical and social challenges along the way," *Information and Software technology*, vol. 57, pp. 21–31, 2015.

[15] H. H. Olsson and J. Bosch, “From opinions to data-driven software r&d: A multi-case study on how to close the’open loop’problem,” in *Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on*. IEEE, 2014, pp. 9–16.

[16] B. Plattner and J. Nievergelt, “Special feature: Monitoring program execution: A survey,” *Computer*, no. 11, pp. 76–93, 1981.

[17] H. van der Schuur, S. Jansen, and S. Brinkkemper, “A reference framework for utilization of software operation knowledge,” in *Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on*. IEEE, 2010, pp. 245–254.

[18] H. H. Olsson and J. Bosch, “Post-deployment data collection in software-intensive embedded products,” in *Continuous Software Engineering*. Springer, 2014, pp. 143–154.

[19] A. Fabijan, H. H. Olsson, and J. Bosch, “Customer feedback and data collection techniques in software r&d: a literature review,” in *Software Business*. Springer, 2015, pp. 139–153.

[20] N. Delgado, A. Gates, and S. Roach, “A taxonomy and catalog of runtime software-fault monitoring tools,” *Software Engineering, IEEE Transactions on*, vol. 30, no. 12, pp. 859–872, Dec 2004.

[21] A. Ebnenasir, “Designing run-time fault-tolerance using dynamic updates,” in *Proceedings of the 2007 International Workshop on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Computer Society, 2007, p. 15.

[22] E. Ries, *The lean startup: How today’s entrepreneurs use continuous innovation to create radically successful businesses*. Crown Books, 2011.

[23] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, and H. Liu, “Data warehousing and analytics infrastructure at facebook,” in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 1013–1020.

[24] J. Bosch, “Building products as innovation experiment systems,” in *Software Business*. Springer, 2012, pp. 27–39.

PUBLICATION V

Objectives and Challenges of the Utilization of User-Interaction Data in Software Development

Suonsyrjä S., Sievi-Korte O., Systä K., Kilamo T., Mikkonen T.

In the Proceedings of the 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) 2018. pp. 357-361. IEEE.

Objectives and Challenges of the Utilization of User-Interaction Data in Software Development

Sampo Suonsyrjä, Outi Sievi-Korte, Kari Systä, Terhi Kilamo
Tampere University of Technology
P.O.Box 527, FI-33101 Tampere, Finland
{sampo.suonsyrja, outi.sievi-korte, kari.systa, terhi.kilamo}@tut.fi

Tommi Mikkonen
University of Helsinki
P.O.Box 68, FI-00014 Helsinki, Finland
tommi.mikkonen@helsinki.fi

***Abstract*—Understanding users, their requirements and usage patterns helps building better software. To continuously improve operations, user-interaction (U-I) data can provide developers with interesting new possibilities. This study aims at gaining an understanding of how software teams can start the use of U-I data and what challenges they face with it. In this paper, we describe three cases from different organizations. This includes explaining the activities, objectives, and challenges of each team in their efforts to begin using U-I data. We conducted the study with Action Design Research (ADR) method, resulting in findings from each case by intervening in the work of these teams. As a contribution, we designed a U-I data utilization method that summarizes teams' activities. Secondly, we refined and validated the categorizations of U-I data analysis and utilization objectives, and finally we categorized the challenges that the case teams faced. Together, these contributions lay out clear steps also for other software teams in starting the utilization of U-I data.**

I. Introduction

The use of data is increasing in all fields of business and data has been regarded even as "the main driver for innovation" [1]. Consequently, also data-driven software engineering is trending with organizations such as Google, Facebook, Microsoft, and IBM leading the way [2]. For software teams, an interesting option in this field is to collect user-interaction (U-I) data by tracking the actions of users in the user-interface level. For example, the number of times a certain button is pressed during a use session produces such U-I data. With proper analysis, these data provide concrete touch-points for software teams to what the users are actually doing with the software [3].

However, for teams who are only beginning the use of U-I data, it can be problematic to even understand what kind of objectives they could have for the collection and analysis of U-I data. For advancing the use of U-I data, a one size fits all guidance to software teams is difficult to come by, even if the emergence of general tools, such as Hotjar[1] and Google Analytics[2], have simplified the collection and analysis of data especially in the field of web development. For installable applications, the common solutions seem to focus on specific tasks such as Apache Log4j 2[3] on logging. In addition, scientific models such as the data dimension of the Stairway to Heaven [4] or the HYPEX model [5] can provide good starting point for understanding the field. However, there seems to be a lack of understanding of how organizations concretely start the utilization of U-I data and what kind of challenges are related with it. To fill the gap, we study

1) what kind of activities software teams do when starting the utilization of U-I data,
2) what kind of objectives software teams have for U-I data,
3) and what kind of challenges software teams face after starting the utilization of U-I data.

The study is a part of a large research effort, where the aim is to support data-driven software development. Previously, we have studied three case organizations (A, B & C) on how software teams choose collecting techniques for U-I data [6]. This time, we intervene with the work of the same software teams in order to start the actual use of U-I data. As contributions, (1) we design a U-I data utilization method to model and guide the workflow of how software teams start the use of U-I data. Then, (2) we examine the categorizations of U-I data analysis and utilization objectives from our previous work [7] in practice. Finally (3), we categorize the typical challenges with the utilization of U-I data.

II. Background and Related Work

Data from the Use of Software: Many terms are used to describe the data that is gathered from a software system. The data types for collected data vary by their nature in respect to time or type that is generated. One category of software data is *runtime data*, which consists of product data gathered as the system is running. Tightening this, Holmström Olsson and Bosch [8] have used the term post-deployment data for data that are generated by a product after its commercial deployment. Barik et al. [9] have approached software data somewhat similarly by events. They split this event data into two categories based on the behavior of the data, namely logs and telemetry. U-I data, as used in this paper, can be categorized as telemetry with the specification that it originates from user-interactions. Shuur et al. [10] have coined the term "software operation knowledge" and categorized data types related to SOK into usage (e.g. U-I), performance, quality and feedback data.

Uses for Data in Software Development: In the same study, Schuur et al. [10] have defined a framework that

[1]https://www.hotjar.com
[2]www.google.com/analytics
[3]https://logging.apache.org/log4j/2.x/index.html

lists uses for software operation knowledge. Similarly, in our previous work [7], we categorized uses for post-deployment data and analyses that software teams could conduct on post-deployment data based on a questionnaire survey. In a bit more general level, Begel and Zimmerman [11] collected an extensive 145 question list of questions that software developers would like to have answers to in software development. The question of ”How do users typically use my application?” was ranked essential the most often in their survey.

Studies by Rodriguez et al. [2] and Sauvola et al. [3] investigated data collection practices and how monitoring customer usage scenarios, A/B testing and other experiment tools can be used to collect useful feedback and help with understanding of customer expectations. Also a larger literature review for customer feedback collection techniques in software was done by Fabijan et al. [12].

While those studies focus on data collection and its challenges, additional studies by Roehm et al. [13] and Guzman et al. [14] focus more on the methods that are used to analyze the data and how this can be used to enact improvements on the software product. The research by Guzman et al. has culminated to the Q-Rapids framework, where quality and functional requirements are managed together, evaluated incrementally and runtime data from the system is aggregated into indicators that can be used to support decision making in further development cycles. Somewhat similarly, the HYPEX model by Olsson et al. [5] depicts how data from the use of software systems can be not only taken into account in software development but also how to make such data a driver for the process. In an even more concrete manner, Kohavi et al. [15] have created a practical guideline on how to use the different data types gained. However, their work is limited to the web context specializing on controlled experiments such as A/B testing. Bosch [4] investigated the different levels of data use in software development. The data dimension of the Stairway to Heaven model has been extended to also include project management variables and it also includes customer satisfaction, project throughput, revenue and such. In this study, we focus specifically on U-I data.

III. Findings

A. The U-I Data Utilization Method

We conducted this part of the study with the Action Design Research (ADR) method [16] that emphasizes the joint effort of academics and practitioners. We intervened with the work of the teams to start the utilization of U-I data from February 2016 to February 2017. During this time, we collected research data that consist mainly of workshop memos written down by the first author and/or the team members. These included plans for how, why and what U-I data would be collected. We recognized that all the teams implemented very similar activities and we illustrate these with the designed U-I data utilization method. The U-I Data Utilization method is formed of three steps, each consisting of two to three activities as seen in Table I. The table also includes descriptions of how the activities were executed in the three cases. The three steps frame their activities both time-wise and in the sense of how the activities are related to each other. The steps and the activities are described as follows.

The first phase of the method is the **Proof-of-Concept** step. Two evident points in this step are to first *select the collecting technique* and to then *test the selected collecting technique* so it can be integrated with the intended target software. However, there can be other reasons for producing such a prototype as well, such as for internal marketing of the U-I data collecting or to get an approval for it from other parts of the organization. Another part in this phase is to *brainstorm the analysis and use objectives* for U-I data. This initial idea generation can be important also in the selecting of the collection technique. However, the brainstorming can take place multiple times inside this step, both before and after the technique selecting.

The second step in the U-I data utilization method is to **Aim & Deploy**. The brainstorming activity of the first step should result in an excess number of use and analysis objectives, and in most cases all of these cannot be collected in the end. This is due to multiple factors such as limitations in collecting techniques and U-I data analyzing skills of the team members. Based on these factors, the *selecting of the analysis and use objectives* is required before *concretely defining the U-I data collecting points*. The collecting technique and objective selections can affect this defining significantly. Therefore, this activity should be taken into account already in the selection of the collecting technique.

The final phase of the method is to **Collect & Analyze** the U-I data. The activities in this step – *collecting the data*, *analyzing the data with the selected objectives* and *presenting the results for use in the selected objectives* – vary greatly depending on the selections made until this step. Therefore, as our main concern of this step we point out that the capabilities of the team to do these activities and the characteristics of their software project should be taken into account much before coming to this step. As future work, we want to do research on these capabilities. However, based on the findings of this study it seems that these range from things such as how much time the team has available for the U-I data analyzing to their skills and tools of doing that analysis, and from the length of their deployment pipeline to their access to collect U-I data either freely from any actual end-user or restrictively from substitute user-groups.

B. Objectives for the Analysis and Use of U-I Data

For validating and refining the categorization of U-I data analysis and use objectives identified originally in [7], we used a directed approach to qualitative content analysis. The goal of such analyses is to validate or extend conceptually a theoretical framework or theory as described in [17].

1) Analysis Objectives: The analysis objectives found in cases are summarized in Table II. The categories are sorted according to the rank of the occurrence of the objectives. Analysis objectives targeting at evaluation of features were the most frequently selected among their categories. The analysis categories are the following.

TABLE I
STEPS AND ACTIVITIES OF THE U-I DATA UTILIZATION METHOD AND DESCRIPTIONS OF THEM IN CASES

Activity	Case A	Case B	Case C
Step1: Proof-of-Concept			
1: Brainstorming the Analysis and Use Objectives	Internal session led by a UX specialist in May 2016 resulting in a list of 46 questions	Workshop in March 2016 with the first author and two members of the software team resulting in eight objectives for U-I data	Internal session in October 2016 led by the software architect resulting with five objectives
2: Select a Collecting Technique	Monitoring the execution environment of the team's software system with a tool developed by the 1st author	Manual instrumentation in combination with an automated collecting tool	Aspect-oriented programming
3: Test the Collecting Technique	Additional demonstrative application developed and presented in a cross-team demo show	Testing with the local development environment	Testing with the local development environment
Step2: Aim & Deploy			
4: Select the Analysis and Use Objectives	23 questions selected by the first author and three team members	Three features selected for monitoring, each with two parallel implementations	No exclusions of initial questions
5: Define the Collecting Points	Tool configured by the first author and one team member. Merged to the code base.	Code snippets for monitoring the few places attached by one team member & 1st author	Data defined to be collected in selected U-I events by team member and 1st author, collecting snippets integrated by the team member
Step3: Collect & Analyze			
6: Collect Data	Company internal testing day in December 2016. Data sent over HTTP to a MySQL database.	Two-week collecting period in April-May 2016. Data sent to automated collecting tool.	Collecting period in February 2017. Data logged to users' local computers and then sent to team via email.
7: Analysis with the Objectives	Data wrangling in Rstudio and spreadsheet program. E.g. session time calculations	Usage statistics ready in automated tool	Data wrangling in Rstudio and spreadsheet program. E.g. use path visualization
8: Present the Results and Use within the Selected Objectives	Workshop session in February 2017	Presentation to all team members in May 2016	Presentation sessions to a team member in March 2017

TABLE II
ANALYSIS OBJECTIVES IN THE CASES

Analysis Objectives	*Initially*	*Selected*
Value Analysis	27	16
Pain Point Analysis	14	8
Use Paths	13	6
User Profiling	5	1
Total	59	30

TABLE III
USE OBJECTIVES IN THE CASES

Use Objectives	*Initially*	*Selected*
UX	39	19
Informed Feature Development	11	7
Resourcing and Prioritizing	5	1
Requirements Validation	4	3
Total	59	30

Value Analysis: The objective is to find out how valuable a certain feature or system is. In the most simplistic case this is done by calculating how many times the feature has been used in a certain time period and using that information in forming an estimate of the value of the feature. In addition to how often a feature is used, value evaluation can be based also on who is using, where from and at what time. Example from the internal brainstorming session's memo of case A: *"What are the features that are most commonly search for/selected?"*

Pain Point Analysis: The objective is to shed light on possible problems that users might face. These can be related to system performance, low usability etc. Data collected for these analyses likely include timestamps so that the time can be calculated and compared with others or with what was expected. Although not strictly U-I data, collecting error logs is similarly done likely to respond to this objective. Example from case A: *"How long time does it take to save a template?"*

Use Paths: The objective is to find out the order of the use of features. The commonality and abnormality of individual paths can be analyzed or the analysis can focus simply on what is the order on average. For example, a question in case C was formed as *"Use of library: What is done and what is the order?"*

User Profiling: The objective is to segment users based on the U-I data that is collected. Differentiating can be done in many ways such as expertise of user (novice and expert) or technology they use the system with (browser). For example, one analysis objective in case A was: *"What browser and browser version does the user have?"*

2) Use Objectives: The second category of objectives focuses on the different uses that the case teams had for U-I data. Table III summarizes the occurrences of objectives by each category and displays also how many of the objectives were selected for the actual collecting and analysis in the cases. User eXperience (UX) was the most popular use objective category. The use categories are the following.

Informed Feature Development: The objective is to get insights to support decision making about how or whether features should be developed (further). The insights can include pointers to which features require improvement and guidelines for new designs. These are then to be used for making informed decisions in the daily work of the software developers. For example, the team in case C targeted the ”use path” analysis (mentioned as the example above) at finding out if they could develop a new feature called ”recently opened” to their library functionality.

Requirements Validation: Requirements for a system might come from other stakeholders than users. The objective highlights this by aiming to validate the requirements with the users. For instance, the intention for analyzing which browsers are used (i.e. the example from the user profiling category) was to know how many browsers the software should support. For existing features these objectives can start off easily by collecting usage information, but for non-existing features approaches such as creating a minimum viable product [18] (or rather a minimum viable feature) need to be applied.

Resourcing and prioritizing: The objective is to use results for prioritizing or resourcing related decision making. Such results could be formed in a feature's value analysis, which would then affect the priority of the related development tasks in the products backlog. Similarly, additional development staff or server space could be resourced towards the high-valued features. For example, the above value analysis example from case A could work as a basis for prioritizing the development work of certain features.

UX: The objectives of this category are similar to the ones in the informed feature development category but they focus specifically on the system's User eXperience (UX). For example, a question formed as *“Does the user open the Help?”* from the case C was targeted at understanding in which situations the users miss help.

C. Challenges in the Utilization of U-I Data

We interviewed the case teams in January 2018 after a year from the initial U-I data utilizations. For analysis, we used conventional content analysis [17] to form categories of the challenges the teams faced with U-I data utilization. We found three **themes** of challenges that all of the case teams had faced after a year from starting the utilization of U-I data. These divide into *sub-themes* that characterize the main themes. In the following, we describe the themes and present translated quotes from the interviews accordingly.

Team members had **value concerns** of U-I data utilization in the sense that they thought it had either *low value* or *unclear value*. UX Specialist in Case B: ”I guess, its value is not seen here so high that someone else than me would start doing it.” Seeing the benefit and value under the bottom line seemed especially difficult, which then can result in other challenges such as getting organizational permissions. That again, makes the challenges in this theme significant.

The interviewees had also faced challenges related to the **difficulties in U-I data utilization**. Sub-themes of *Technical concerns and difficulties, Difficulties in the extraction and or use of insights, High effort required for U-I data utilization, Scalability concerns for U-I data utilization, Lack of experience in U-I data utilization, Using Unspecific Objectives* characterize these challenges. For example, the software architect in case C mentioned how ”In this context the experience was that although we got some results out of the analysis, it was still pretty difficult to draw conclusions.”

Finally, the team members brought up challenges related to the **unsuitability of U-I data utilization in the current situation**. Some interviewees felt they were faced with *lack of resources* or *lack of support from organization*. They also reported on how U-I data utilizing had *conflicts with ways of working, methods, technical environment and/or culture*. For example, the operational development manager in case A described such conflicts: ”If our product was in the Internet world we would have implemented this already. But you see this is not a service but a product there behind firewalls”. Additionally, the interviewees had *Privacy Concerns* as well as *Legal Concerns*.

IV. Discussion and Threats to Validity

We designed the U-I data utilization method to guide software teams that are transforming or simply trying out more data-driven software development methods. It combines the activities we identified in the cases of this study and lays out practical steps for other software teams who have similar intentions. There are some similar methods in the scientific literature. For example, a framework presented by van der Schuur et al. in [10] describes software operation knowledge and its identification, acquisition, integration, presentation and utilization in software engineering. The framework seems data centric in the sense that it is designed to reveal the potential role of software operation knowledge. At the same time, it lacks the concreteness that would help practitioners in starting the use of such knowledge. However, it takes into account also the perspectives of the company and customer and not only of the development as is in our study. Given the widening field of tasks that especially agile software teams are responsible for, the utilization objectives from the company and customer perspectives might become increasingly important for the software teams. On the other hand, the HYPEX model [5] is also highly specific with its objective on how to integrate the use of data, e.g. U-I, into software development process. Compared to our U-I data utilization method, the HYPEX model requires a radical change in the software development process whereas the method designed in this study rather focuses on supporting the existing software processes.

Considering the use and analysis objective categories, we found that the *UX* related use objectives and *Value Analysis* related analysis objectives were selected the most. Many of the objectives in the *Value Analysis* category are quite simple to approach when compared to objectives in categories such as *Use Paths*. Similarly, the objectives in *UX* category likely are more concrete than the objectives e.g. in the *Resourcing and Prioritizing (R&P)*, and they do not necessarily need other information to support decision making as is the case with *R&P*. In the three cases, only team B had experience in collecting U-I data. In this sense, it seems quite natural how all of the teams decided to start by trying out these simpler tasks even if they identified various other objectives in the first brainstorming sessions. Moreover, we think that with the extended descriptions and lively quotes from the cases, the refined analysis and use objective categories can provide software teams with helpful insights and inspiration about what to do with U-I data.

Within this research effort, we had a great opportunity to study the same cases after one year from the initial U-I data utilizations. This resulted in an in-depth look to the challenges the case teams had faced during the year. What was similar among all the cases, was that all of them had faced the challenges related to the unsuitability and difficulty of U-I data utilization. Each of them also had concerns about its value. Considering the inspirational success stories such as [18], the reports on low or unclear value of data utilization sound alarming. However, the other two challenge themes found in this study can explain this. For example, in some cases the technical environment as a whole requires radical changes to make U-I data utilization possible. The changes might be required also in areas that are not in the control of the software team. Such challenges were categorized as process and customer immaturities in [7].

Considering validity, we point out that while the number of case teams was low (3), the case organizations were highly different in their characteristics among each other. In this sense, the results of this study provide a detailed look on the activities, objectives, and challenges of software teams in the utilization of U-I data. Therefore, the results can give interesting insights to a wide audience even if the results of this study would not be generalizable for any software team. The bias from using only one researcher for most of the work in this kind of a study is significant. To lower this, the meeting notes were shown to and accepted by team members in each case. Moreover, the second, third, and fourth authors participated in the analysis of the research data. Additionally, the second author, who participated in the final interviews and in the analysis parts of this study, did not participate in the ADR part.

V. Conclusions

This study aimed at gaining an understanding of how software teams can start the use of U-I data and what challenges they face with it. In this paper, we described three cases from different organizations by explaining how the teams identified analysis and use objectives for U-I data, selected collecting techniques, collected the data and analyzed the results. We conducted the study with ADR, and as a contribution designed a U-I data utilization method that summarizes the steps of the case teams in their efforts of starting the use of U-I data. In addition, we validated and refined the categories of U-I data analysis and use objectives we had found in our previous work.

In addition, we studied the challenges software teams can face while starting the utilization of U-I data. The results contribute to the practice in the sense that the presented U-I data utilization method lays out steps for other software teams to try similar processes. At the same time, the found challenges can give valuable consideration points while the use and analysis objectives work as inspiration.

References

[1] H. H. Olsson, J. Bosch, and H. Alahyari, “Towards r&d as innovation experiment systems: A framework for moving beyond agile software development,” in *Proceedings of the IASTED*, 2013, pp. 798–805.

[2] P. Rodriguez, A. Haghighatkhah, L. E. Lwakatare, S. Teppola, T. Suomalainen, J. Eskeli, T. Karvonen, P. Kuvaja, J. M. Verner, and M. Oivo, “Continuous deployment of software intensive products and services: A systematic mapping study,” *Journal of Systems and Software*, vol. 123, pp. 263–291, 2017.

[3] T. Sauvola, L. E. Lwakatare, T. Karvonen, P. Kuvaja, H. H. Olsson, J. Bosch, and M. Oivo, “Towards customer-centric software development: a multiple-case study,” in *Software Engineering and Advanced Applications (SEAA), 2015 41st Euromicro Conference on*. IEEE, 2015, pp. 9–17.

[4] J. Bosch, *Speed, Data, and Ecosystems: Excelling in a Software-Driven World*. CRC Press, 2017.

[5] H. H. Olsson and J. Bosch, “From opinions to data-driven software r&d: a multi-case study on how to close the’open loop’problem,” in *Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on*. IEEE, 2014, pp. 9–16.

[6] S. Suonsyrjä, “Eeny, meeny, miny, mo... a multiple case study on selecting a technique for user-interaction data collecting,” in *International Conference on Agile Software Development*. Springer, 2017, p. To appear.

[7] S. Suonsyrjä, L. Hokkanen, H. Terho, K. Systä, and T. Mikkonen, “Post-deployment data: A recipe for satisfying knowledge needs in software development?” in *Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2016 Joint Conference of the International Workshop on*. IEEE, 2016, pp. 139–147.

[8] H. H. Olsson and J. Bosch, “Post-deployment data collection in software-intensive embedded products,” in *Continuous software engineering*. Springer, 2014, pp. 143–154.

[9] T. Barik, R. DeLine, S. Drucker, and D. Fisher, “The bones of the system: a case study of logging and telemetry at microsoft,” in *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM, 2016, pp. 92–101.

[10] H. van der Schuur, S. Jansen, and S. Brinkkemper, “A reference framework for utilization of software operation knowledge,” in *Software Engineering and Advanced Applications (SEAA), 36th EUROMICRO Conference on*. IEEE, 2010, pp. 245–254.

[11] A. Begel and T. Zimmermann, “Analyze this! 145 questions for data scientists in software engineering,” in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 12–23.

[12] A. Fabijan, H. H. Olsson, and J. Bosch, “Customer feedback and data collection techniques in software r&d: a literature review,” in *International Conference of Software Business*. Springer, 2015, pp. 139–153.

[13] T. Roehm, B. Bruegge, T.-M. Hesse, and B. Paech, “Towards identification of software improvements and specification updates by comparing monitored and specified end-user behavior,” in *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*. IEEE, 2013, pp. 464–467.

[14] L. Guzmán, M. Oriol, P. Rodríguez, X. Franch, A. Jedlitschka, and M. Oivo, “How can quality awareness support rapid software development?–a research preview,” in *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2017, pp. 167–173.

[15] R. Kohavi, R. Longbotham, D. Sommerfield, and R. M. Henne, “Controlled experiments on the web: survey and practical guide,” *Data mining and knowledge discovery*, vol. 18, no. 1, pp. 140–181, 2009.

[16] M. K. Sein, O. Henfridsson, S. Purao, M. Rossi, and R. Lindgren, “Action design research,” *MIS quarterly*, pp. 37–56, 2011.

[17] H.-F. Hsieh and S. E. Shannon, “Three approaches to qualitative content analysis,” *Qualitative health research*, vol. 15, no. 9, pp. 1277–1288, 2005.

[18] E. Ries, *The lean startup: How today’s entrepreneurs use continuous innovation to create radically successful businesses*. Crown Books, 2011.

PUBLICATION VI

Understanding the Relations between Iterative Cycles in Software Engineering

Terho H., Suonsyrjä S., Systä K., Mikkonen T.

In Proceedings of the 50th Hawaii International Conference on System Sciences (HICSS) 2017.

Understanding the Relations Between Iterative Cycles in Software Engineering

Henri Terho, Sampo Suonsyrjä, Kari Systä and Tommi Mikkonen
Tampere University of Technology, Tampere, Finland
{henri.terho, sampo.suonsyrja, kari.systa, tommi.mikkonen}@tut.fi

Abstract

Iterations are one of the most successful mechanisms in software development to ensure that the resulting system is satisfactory. Due to its strengths, various kinds of iterations have been integrated to software development with varying goals. In this paper, we consider different types of iterations related to software development, *including prototyping, incremental development, sprints as in e.g. Scrum, and iterations* as defined in *Lean Startup. The goal is to understand the relations between the types of iterations, and to find out what kind of similarities and differences they have with each other. As a result, we find that while the goals are different, it is possible for the iterations to coexist, so that one form of iteration is used as a tool to complete the goals of another.*

1. Introduction

While often considered as a modern approach compared to plan-driven, waterfall-style approaches, iterative development has a long history – the application of iterative and incremental development dates as far back as the mid-1950s [1]. While no single iterative approach was dominant and numerous differences between methods existed, they all shared the view to avoid a single-pass, sequential, document-driven, gated-step process [1].

Different iterative methods and techniques for different phases of software development have been proposed by the software engineering community. For example, prototyping [2], Scrum [3] and, more recently, Lean Startup [4] share an iterative way of working. However, these techniques have born from different viewpoints, and they are to be used at different stages and for different purposes in the development process. For instance, while sprints are used to manage weekly tasks [3], Lean Startup is used to test initial product viability [4].

Since the term iteration is used in so many contexts and meanings, ranging from a minimum viable product that can be used to test business hypothesis to full-blown new versions of software products, it is not surprising that the overlapping use of methods can cause confusion in the process. The situation is further complicated by the fact that numerous stakeholders, with different terminology but partly the same terms, often participate in software development activities in different roles, such as customer, domain specialist, project and product manager and developer, to name some common ones.

The communication problems between the stakeholders of the software development process are a major issue in software development. The different goals of different stakeholders can result in conflicts between priorities even though all are in their own opinion speaking the same language. These problems are exacerbated in large organizations, where communication between stakeholders is already a larger issue in its own. If the knowledge of the different types of iterations and their targets, attributes, and stakeholders would be improved, the strengths of all the cycles could be better utilized. This in turn would lead to more integral working between projects and organizations, and creating common tools and vocabulary to the whole development team.

Some authors claim that in the end, the cycles culminate in running code that is continuously maintained [5], but we assume a wider view. We claim that iterations also serve other purposes, and that iterations proposed by different approaches are inter-related but not the same. We believe that when understood properly, these different cycles could actually result in better overall view of the product development and communication between the different stakeholders in software development. This better view can be utilized to optimize the usage of resources, understand feedback better and make better decisions on the development track of the project as a whole, resulting in higher quality products.

In this paper we analyze similarities, differences, and other relations between the different forms of iterations used in software development. The paper has been inspired by earlier work regarding how software startups handled product development [6]. Extending this work to cover the different types of iterations instead of simply product strategy introduces more possibilities to apply the results in practice.

The rest of this paper is structured as follows. In Section 2 we introduce the iterations in the selected approaches together with a brief comparison of their characteristics, goals, and motivations. In Section 3, we shift the focus to the various targets of software development cycles. In Section 4, we discuss the stakeholders of software development, and in Section 5 we address the attributes of the various cycles. In Section 6, we provide a synthesis of the results. In Section 7 we draw some final conclusions.

2. Background and Related work

The researchers and practitioners of software engineering have introduced several ways to iterate in software development. These different types of iterations are used in different context but have several similarities. Managing these iterations takes work and specific attention, as well as balancing between time to market [7].

As also discussed by Berente and Lyytinen, iteration is actually a multi-dimensional issue where different levels of iteration always happens in a software project, be it cognitive or guided by the process [8]. In this paper, however, we focus more on the different forms of iterations as methods and on their various characteristics as such. More precisely, we analyze similarities, differences and other relations between the different forms of iteration in four different setups. The analyzed iteration types are the following:

- **Prototyping**. Prototypes enable a high degree of user evaluation and initiates a learning process for the end users and developers of the system [2].
- **Incremental development**. The features of the software are grouped so that the most important features are implemented first, and the subsequent iterations complement the software [9].
- **Sprint**. Popularized by Scrum [3], sprints contain time-boxed sets of features selected for implementation.
- **Lean Startup**. Popularized by Eric Ries, Lean Startup is an iterative development method for creating products that users actually want and are ready to pay for [4].

As the starting point of our study, we next briefly review the different iteration types together with the drivers behind these approaches.

2.1. Prototyping

Software development approaches that are based on prototyping have been developed for situations where the work steps of a project cannot be clearly detailed before execution [10]. Prototyping incorporates many styles, including iterative, rapid, evolutionary, throwaway incremental, and mock up prototyping [11]. Stephen and Bates [2] define the prototype through two common characteristics:

1. The prototype enables a high degree of user evaluation, which then substantially affects requirements, specifications, or design.
2. The prototype initiates a learning process for users and developers of the system.

Hierarchically, prototypes can be divided into throwaway and evolutionary prototypes. Throwaway prototypes are discarded after their initial use, but evolutionary prototypes are used as a basis for further development. Thus, development based on evolutionary prototypes goes through sequences of re-design, re-implementation and re-evaluation without knowing the complete set of requirements beforehand [11]. Although the exact requirements for further development might be unclear, the implementation choice still matters as large parts of the code will be reused. In contrast, the intended further use of throwaway prototypes is clear from the beginning – its code will not be used.

2.2. Incremental Development

The *Guide to the Software Engineering Body of Knowledge* defines incremental development as *"An incremental model produces successive increments of working, deliverable software based on partitioning of the software requirements to be implemented in each of the increments. The software requirements may be rigorously controlled, as in a linear model, or there may be some flexibility in revising the software requirements as the software product evolves*." [12].

While incremental development is often considered a somewhat modern technique, Craig Larman and Victor Basili argue that its application dates as far back as the mid-1950s [1]. In incremental development, completed increments are deployed and taken into use. A particular feature of incremental development is that all increments are planned according to the needs of the users and the development gets feedback from the real usage for designing and deploying later increments. Still, a plan over multiple increments to come is commonly made, so that each increment can be used to drive the design towards future requirements. In terms of concrete realizations, RUP introduces four distinct project life cycle phases:

- **Inception**: Scope the system so that there is a valid baseline initial budgeting.
- **Elaboration**: Mitigate the key risks; execute problem domain analysis; define baseline architecture.
- **Construction**: Build the system.
- **Transition**: Take the system to production.

While it is possible to advance in iterative cycles within each phase, the above phases, when repeated, form the incremental development cycle as defined by RUP. Consequently, each cycle is planned almost to the extent as a one-off product would be planned, thus resembling the waterfall model. However, it is common that some of the features are pushed to subsequent releases, in particular if they do not contain near-term value for end users.

2.3. Sprints in Agile Methods

Two core values in the Agile manifesto are "Customer collaboration over contract negotiation" and "Responding to change over following a plan" [13]. For obvious reasons, these values conflict with rigorous control and up-front definition of the requirements that are often associated with development methods where a longer-term view is used, or, in general, with any precise interpretation of a pre-made project plan.

Many concrete incarnations of agile methods, for instance XP and Scrum, include time-boxed sprints where technical activities take place. The primary purpose of these sprints is coordination and management of work. Furthermore, Scrum proposes the production of potentially deliverable products in all sprints, where the content of each sprint is usually defined according to development and customer collaboration aspects – not based on the incremental needs of a user.

2.4. Lean Startup

Lean Startup is a methodology for building enterprises, not software [4,13]. The methodology has been crafted in software context and it shares the idea of frequent iterations with many software engineering methods. In a nutshell, building a successful product for a software startup consists of multiple short iterations each of which surveys systematically the context of the conceptualized product.

The iteration in Lean Startup starts with an idea that includes hypotheses about the customer behavior or the context of usage. When the first hypotheses have been validated, the first *minimum viable product* (MVP) can be built. This product is a version that enables a full turn of the build measure learn loop with minimum effort. For each loop the main goal is to learn if the business and product hypothesis is valid – in other words, whether the product is actually something that someone needs or wants and can it create a scalable business.

Based on the above, the goal of the process is to evaluate the business validity of the proposition. However, technological development is required in most cases to do such evaluation, in particular in the context of software development [15].

3. The Changing Targets of Software Development Cycles

Many challenges in software development have led to different kinds of iterations. Firstly, there are many unknowns related to technologies, requirements and business. This means that iterations are needed to *manage risks and learn from feedback*. For complex systems, these challenges exist even if the context of the project is stable. However, the context – customer needs, technologies, and so on. – usually change. Thus iterative approaches are used to *effectively respond to changes*. The fact that the challenges vary in their nature means that we must use iterations for several purposes. In the following, we formulate the types of problems that match the types of iterations addressed in this paper. In addition, the role of iterations within the larger scope of development is addressed.

3.1. Prototyping

The goal of prototyping is probably the most straightforward, when considering the different types of iterations – build a prototype simply to figure out what is doable and what is not. Prototyping is often needed to get started with something new, be it implementation technique or domain. In addition to testing or trying a technology, prototyping is used to communicate ideas to stakeholders.

The value of a prototype is not primarily in the developed software or its use. The value is in the learnings and communication. The developer organization learns from the issues in development, benchmarks and stakeholder feedback. In addition, the prototype helps in communicating the idea or product.

Prototyping methods have a long-standing tradition also in the field of human-computer-interaction (HCI)

[16]. In general, prototypes range from high to low fidelity, i.e. from low-cost methods such as paper sketches to more detailed propositions like interactive web applications. Low-fidelity methods have proven to be highly efficient in validating designs and predicting large problems. On the other hand, high-fidelity methods have been used for example for assuring management and other stakeholders.

While often considered as small experiments, prototypes of considerable size also exist. For instance, the Cloudberry project [17] – obviously beyond a simple, single-developer experiment of a particular detail – can be regarded as a prototype for demonstrating the feasibility of web technologies in the development of a mobile device. In fact, although seldom mentioned explicitly, many totally new software systems can be traced back to prototypes created to test technology, which, when deemed mature and applicable, are eventually refined to products. Clearly, organizing such complex prototypes needs different kinds of iterations to help the development.

3.2. Incremental Development

Almost any computing system we are accustomed to is a result of several evolutionary steps. These steps, reflecting the understanding of user needs at a particular moment, as well as development capabilities available at the time, are used to create a product in such a way that changing technology during the development can be integrated into the process to create simpler, better results which are easier to maintain and develop further.

While often associated with new features introduced in each iteration, it is sometimes in the eye of the beholder how much iterations have in common. For instance, while one can consider the different Microsoft Windows versions as increments, it is questionable to what extent the different iterations share their code base. Thus, incremental development can be considered from various angles, one angle considers the technical origins, and others focus on the development organization and end users of the system. While the last angle is often overlooked, keeping customer happy with new and improved features is an important part of incremental development – indeed, if no new versions emerge, the users may think that the development has ended and there is no maintenance left, encouraging them to start using another, competing system.

3.3. Sprints

Sprints as understood in Scrum [18] most likely have the most concrete, unambiguous definition of any cycle in software engineering. Simply put, sprints are time-boxed, repeated cycles during which software development takes place. Each cycle contains a number of events, such as Daily Scrum and retrospective, which help in execution and coordination of the work, as well as enable improving the ways of working. Thus, sprints can first and foremost be considered as a way to organize software development, and to associate the work with fixed starting and ending points. What happens during the sprint is up to the Scrum team that can independently decide how to meet the targets of the sprint. Since the focus and commitment is on one sprint at a time, the team can respond to change only in the next sprint. However, since the sprints are usually short, between 2-4 weeks, it is usually enough to shift the focus to next tasks only in the next sprint.

Based on the above, sprints can be regarded as a project management mechanism for the development. Advancing in increments enables frequent evaluations as well as forces the developers to verify and validate the system each time a sprint terminates, making it a solid starting point for the development.

3.4. Lean Startup

In Lean Startup, iterations consist of three phases – build, measure, and learn as illustrated in Figure 1. Each phase plays a role in gathering justifiable evidence if profitable, scalable user needs exist – and what is a feasible business model or a product to fit to the model. The goals of the phases are presented in the following:

- **Build**: Create the simplest possible version of the system that fulfills the intended mission of the system, based on hypothesis of the users need.
- **Measure**: Collect data from the use of the system, preferably so that it gives statistically significant evidence that either validates or rejects the hypothesis.

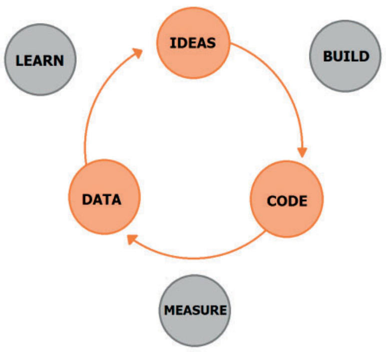


Figure 1 Build Measure Learn Cycle

- **Learn**: Based on measurements, determine whether or not the mission was accomplished in accordance to the hypothesis. If the mission was not accomplished, redefine the hypothesis and initiate a new build measure learn cycle.

It is important to notice that while software may be built as a part of executing the Lean Startup process, its goal is to validate a business hypothesis, not to be a full-fledged product. Hence, the notion of Minimum Viable Product (MVP) is used to denote a version of the system that includes enough elements to judge its business potential, but which by no means is a complete product.

3.5. Summary

The targets of the different cycles are presented in Table 1 and briefly summarized in the following.

For prototyping, the main focus lies in turning ideas, thoughts, and intuitive designs into something concrete. The target is communication: either to get feedback or to communicate the idea to external stakeholders. This is achieved by turning ideas, thoughts, and intuitive designs into something concrete. Although the produced solutions can be small and cover only one perspective, prototyping is a great way to take the first steps towards the final product.

The main target of Lean Startup's build measure learn loop is to learn by creating something concrete and validating the learning with a specified audience. In contrast to prototyping, the context of learning is business driven although metrics such as amount of new users can be seen software driven as well.

However, both incremental development and sprints emphasize the software and its production. In the incremental development new version are delivered to users one after another and in extreme cases the software development is seen as a continuous flow of new software versions. With such premise, the software team can take advantage of new emerging technologies that become available during the software development. On the other hand, the team can also respond to the changing user needs faster and easier than with more traditional methods.

Although sprints might guide the software teams into the same kind of benefits as incremental development, one of their core targets is to freeze at least some parts of the user needs and requirements. In this sense, sprints help the teams in execution and coordination of the work by providing time-boxed

Table 1 Targets and Attributes of the Cycles

Cycle	Targets	Attributes
Prototyping	• Figuring out what is technically doable • Validating designs and predicting large problems • Communication, assuring management and other stakeholders	• Cycle length: From hours to months • Team size: From one developer to a team of developers • Termination condition: Full stop once a technological solution is proven to be feasible.
Incremental development	• Provide value to the customers already during the project. • Taking advantage of new technology • Assuring the stakeholders that the development is continuous and on-going	• Cycle length: Any given time that is needed to get a new increment done • Team size: Software team (and the related stakeholders) • Termination condition: When the new software asset / increment is considered done.
Sprints	• Responding to emerging user needs • Helping in execution and coordination of the work • Improving the ways of working • Guiding to frequent evaluations of new parts of the system	• Cycle length: Evenly one to four weeks • Team size: Software team • Termination condition: Calendar deadline
Lean Startup	• Gathering justifiable evidence if profitable, scalable user needs exist. • Evaluating if a hypothesized business model is feasible to satisfy the user needs. • Learning by creating MVPs.	• Cycle length: From days to weeks • Team size: From a single developer to a whole software team. • Termination condition: Once the learning goal can be validated with statistically significant results.

segments with clear targets.

4. Role of Stakeholders

Almost all software development projects involve various stakeholders. At least the following roles are commonly identified:

- **Individual developers** that participate in the development in different roles, like designer, programmer, and tester. Together, they form the development team, which can sometimes be considered as a separate stakeholder as well.
- **End-users** are the individuals and organizations that eventually use the designed software system. Most commonly the developers and the end-users have different backgrounds and therefore have a different view to the system.
- **Customers** represent the organization that make the investment decision, provide the requirements and decide if the software system is to be taken to use. The relation between end-users and customers is often overlapping – you first buy a system, and then you use it – but at times the roles are distinct.
- **Sponsors** are investors that help development team to start their work, when a paying customer is still to be found or if the current revenue stream does not yet cover the development costs.
- **Software organization** provides support for the developers. For instance, they may provide support for product management, marketing, sales, and number of other things that fall beyond the actual development. Obviously, each specialized actor inside an organization can be considered as yet another stakeholder, but for the purposes of this paper, they can all be treated similarly.

Stakeholders of the different iterative software development cycles are described in the following subsections and summarized in Table 2.

4.1. Prototyping

Prototyping involves several stakeholders. Prototypes may be used to collect feedback from any of the above stakeholders. End-users and customers can give feedback on usability and feature set of the developed product. Sponsors and organization can give feedback about profitability and other business aspect.

In addition, prototypes are used to communicate the content of the designs and to gain commitment from any of the stakeholders. Based on the information the stakeholders can plan their own activities and increase their interest and trust in developed software and the development team.

4.2. Incremental Development

In incremental development a software organization repeats its development activities one

Table 2 Stakeholders of the Cycles Summarized

Cycle	Developers	End-users	Customers	Sponsors	Organization
Prototyping	Learn about the tested topic	Get early information about the forthcoming software	Get early information about the forthcoming software	Get confirmation about the progress	Get early information for supporting actions
Incremental development	Can concentrate on manageable set of tasks. Reduced risk with early feedback.	Early value: can start using SW and features earlier. Can give feedback.	Early value: can start using SW and features earlier. Get information of the progress. Can give feedback.	Get reliable information of the progress.	Get early revenue. Get reliable information of progress.
Sprints	Can Concentrate on manageable set of tasks. Reduced risk through early feedback.	Can give early feedback at the end of each sprint.	Can give early feedback at the end of each sprint.	Can give early feedback at the end of each sprint.	Can give early feedback at the end of each sprint. Ability to change direction due to changed business situation.
Lean Startup	Get fast feedback to minimize waste	Early value: can start using SW and features earlier. Ability to give feedback.	Get early information about the forthcoming software	Get fast feedback on the business potential.	Get fast feedback on the business potential.

round after another. In the case of RUP, these activities include inception, elaboration, construction, and transition that are further decomposed to smaller increments. Also, different kinds of variants can be derived for company-specific use. Feedback from end users, including also usage data collection, as well as marketing and sales can be taken into account as a part of the development, and in general the approach is comprehensive in the sense that it involves almost any possible stakeholder of the software, including developers and testers, organizational support functions, as well as end users and customers.

The overwhelming range of interest groups makes it sometimes difficult to determine all the consequences the introduction of a new version produces. Obviously, phasing of the project means that the set of involved stakeholders is not the same in each phase. Furthermore, since the different phases in themselves include several activities – such as alpha and beta testing – defining the precise set of stakeholders for the life-cycle is next to impossible as every stakeholder is somehow involved at some point.

4.3. Sprints

Sprints are executed by software teams, so software developers and testers are obvious stakeholders. However, any outside communication with the team takes place via a product owner, who acts as a proxy for all other stakeholders. Therefore, the number of stakeholders in the middle of sprints remains low. However, after each sprint, feedback from stakeholders is requested. Preferably an executable version of the system is then demonstrated to other stakeholders, such as product managers, customers, and end users to gather feedback and foster mutual commitment to the development. In these demonstrations stakeholders both learn about the developed software but also have possibilities to give feedback.

4.4. Lean Startup

As long as the decided end-result of the build measure learn cycle is a software artifact, individual developers are obviously entwined in the loop. However, the software organization is likely the most influential of the stakeholders, because the bottom line target of an MVP is commercial. Consequently, the *software* from the defined *software organization* term above can many times be eased out, because it is not uncommon that the organization for example subcontracts the software development of their MVP.

Although the software organization might be the one calling the last shots when building an MVP, the influence of potential customers and investors cannot be emphasized enough. As the main idea in the development of an MVP is to get feedback from other stakeholders, refining it towards something that customers want intrinsically requires their input to the subject. Additionally, or in some cases even with the heaviest focus, MVPs can be developed to assure and engage investors.

5. Attributes of Software Development Cycles

To understand the software development cycles and their nature more deeply, we select three dimensions that are continuously present with them. These descriptive dimensions are cycle length. work effort or team size per cycle, and a termination condition for a cycle or how is each cycle validated

5.1. Prototyping

Prototyping can have the shortest length of the development cycles, if the low-fidelity paper sketches are considered – such can be completed with a minimal work effort and team size of only one developer or designer. However, be it paper sketching or technological try-outs, the work efforts of prototyping usually stop at once when the required result is reached. In this sense, the amount of work effort and time can be difficult to define in advance.

On the other hand, prototyping can involve much more of the development organization than just one developer. In these situations, the devoted time and work efforts typically require far more careful planning, i.e. risk management by the organization. This, again, can have an impact on the required result of the prototyping cycle as well, because the decision whether the result is sufficient enough is not for only one person to make. For example, a paper sketch or an experimental design can be done by only one designer. In contrast, when a whole organization is devoted to prototyping whether a technological solution is feasible, opinions on termination conditions are bound to raise debate, thus requiring careful planning.

5.2. Incremental Development

Incremental development relies on well-planned, established process, where each of the phases in the life cycle form a solid basis for the subsequent phase. For instance, only after inception it is possible to start to elaborate the project into an implementation form, and only an elaborated enough project can really result in an implementation. Due to such planning, the life cycle of a RUP project can take considerably long time to run – up to years for each iterations in the case of complex products such as telecommunications systems. Due to the extended period of the life cycle, also the development effort can be considerable, up to 1000 man-years in the case of large systems.

Since each iteration in incremental development produces a real system, the outcome for each release includes almost any possible feedback one can imagine. These include technical data such as code quality measurements, test and bug reports as well as business data such as user evaluations, sales reports, and market research studies. The overwhelming amount of feedback can at times be so extensive that it is difficult to utilize all of in the design and planning of the next version of the system.

Since the time it takes to execute a full project life-cycle may be so long, it is not uncommon that the personnel changes in the course of the project. This in turn calls for a procedure to involve new persons in the project in a planned, controlled fashion.

5.3. Sprints

One of the most important constants in sprints is the stability of the development team, followed closely in importance by the fact that the sprints are always of the same length and executed to the end. The fact that the team works together for extended periods of time results in the ability to create realistic time and work estimates for problems at hand, forming the key enabler to meet the time-boxed deadlines.

5.4. Lean Startup

Although a wide range of artifacts from paper sketches to fund raiser campaigns could be seen as MVPs, we scope this paper to include only MVPs with some sort of technological solutions. Even with this limitation, however, the time and work efforts required in each build measure learn loop can vary quite significantly. On one hand, a landing page describing a product idea and a built-in analytics solution can be made in a matter of hours. On the other hand, a detailed user interface that allows customers to act the same way as is intended with the actual product (but for example the real business logic is still done manually), can take weeks only to build.

The decisive point for the length of the cycle in these situations is the wanted end-result. With the landing page example, the organization has to wait in the measure phase as long as the quantitative data, such as the page visits is statistically significant. With the second example, however, the organization can have a very short measure phase and gather qualitative data from a few specific customers sufficiently to advance to the learn phase.

5.5. Summary

Of the described iterative development cycles, prototyping has he most variable cycles ranging from hours with paper sketching to months spent with more difficult technological evaluations. The team size can vary as well, but once the prototype is evaluated as sufficient, the development with the same learning objective comes to an end. Similarly, cycle times vary in the build measure learn loops with Lean Startup. They are dependent on the set learning goals and therefore on the MVPs under construction. The development time of different MVPs obviously varies case by case, but roughly the times range from days to weeks. Obviously, the different types of MVPs need different amounts of staff to work on them, but typically this amount ranges from a single developer to a software team. If the learning goals are clearly set, the termination condition of a build measure learn loop is clear as well – once the learning is validated.

In contrast to the varying cycle times described above, sprints have a fixed time period, which is usually something between one to four weeks. Incremental development is somewhat similar to this, as it also has fixed goal with which the cycle terminates. However, the needed time depends so heavily on the work effort, that the cycle time varies dramatically from minutes to months or even years. The same obviously applies with the needed work effort and team size.

6. Synthesis

Table 3 presents a summary of the different types of iterations. When considering the focus of the described iterative cycles, prototyping and Lean Startup share a similarity in creating a method for experimentation. However, prototyping distinguishes itself with a clearer focus on feasibility and implementability rather than Lean Startup focusing on the business side. Incremental development and sprints, on the other hand, have their focus more on the way the work is organized - incremental development chopping it feature wise and sprints scoping it in time.

The motivation for using the described cycles clearly distinguish them from each other. Incremental development takes into account a wide mix of background ranging from business reasons to technical aspects and from risk management to evolving customer needs. Sprints, on the other hand, aim to exclude almost all of the aforementioned and liberate developers to focus on only the technical aspects.

Table 3 Summarized Characteristics of the Cycles

Cycle	Focus	Motivation	Goal	Developed by
Prototyping	Feasibility and implementability	Almost always technical in nature	Commonly executed to explore design space for a particular solution.	Can involve an individual developer, or a team of developers if a more complicated system is being explored.
Incremental development	Scoping the technical work feature wise.	A mix between business reasons, technical aspects including risk management, and customer needs.	The goal is to organize company operations as a whole in terms of releases.	Most commonly affects the whole organization, including obviously the developers but also sales, marketing, customer care, and so on.
Sprints	Scoping the technical work time wise.	Mechanism to liberate developers from constant changes to a fixed set of features to implement during the sprint.	Considers mostly development aspects and overlooks others, in particular if following Scrum interpretation.	Traditionally executed by a Scrum team up to 12 people; variations that enable synchronization between different teams exist.
Lean Startup	Learning and experimenting.	Business oriented in nature.	Validate or invalidate a business hypothesis with minimum amount of invested effort.	Usually executed only by a minimal team.

Similar to this, prototyping scopes the development into specific problem solving cases. Lean Startup is something of a mix in this sense, since its motivation is ultimately business oriented, but it surely has to take a wide range of different aspects into account in the end.

Lean Startup and incremental development have a similarity regarding their goals and the people they affect. In both of them, the intention is to scope the development work of the whole organization. The final goal is different, however. With Lean Startup the aim is on validating or invalidating a set business hypothesis with a minimum amount of invested effort and staff - this learning is the ultimate key and the produced software artifact is almost irrelevant. On the other hand, an organization probably does not want to waste any work efforts either with incremental development, but the produced software artifact is the most important thing in this case. Therefore, also the amount of people and different parts of the organization can be a lot greater than with Lean Startup.

With sprints, the goal changes again. Although the produced software artifact is unquestionably of high value, the main intention is to make sure that the defined technical and work management related aspects, such as the amount of people, stay the same during a fixed time period. In a way, prototyping is somewhat of a mix from each of the others. It scopes the work into a specific problem solving case like incremental development, but its main outcome is learning from an experiment as with Lean Startup. In addition, its focus is usually sharply on technical aspects as with using sprints.

7. Conclusions

In this paper we have presented an initial analysis of the different types of iterations. The cycles encompass the whole of product development and its different levels from business planning to product refinement. The higher abstraction level cycles such as Lean Startup and RUP can be achieved using sprints and prototyping. This way the software development process as a whole consists of iterations within iterations producing an interlinking whole that is more than the sum of its parts.

An interesting topic for further research is the developer perspective and psychological aspect of different types of cycles. For example, the motivational aspects of the cycle types may be rather different. In addition, we aim at creating a comprehensive conceptual model that covers the different iterations. With such, we see a lot of potential in industrial collaboration to help us validate the model as well as to test it in practice.

Acknowledgments

The authors wish to thank Need for Speed program funded by TEKES, the Finnish Funding Agency for Innovation, for its support.

References

[1] C. Larman and V. R. Basili, “Iterative and incremental development: A brief history,” Computer, vol. 36, no. 6, pp. 47–56, Jun. 2003. [Online]. Available: http://dx.doi.org/10.1109/MC.2003.1204375

[2] M. Stephens and P. Bates, “Requirements engineering by prototyping: experiences in development of estimating system,” Information and Software Technology, vol. 32, no. 4, pp. 253–257, 1990.

[3] K. Schwaber, “Scrum development process,” in Business Object Design and Implementation. Springer, 1997, pp. 117–134.

[4] E. Ries, The lean startup: How today’s entrepreneurs use continuous innovation to create radically successful businesses. Crown Books, 2011.

[5] T. Mikkonen and K. Systä, “Maximizing product value: Continuous maintenance,” in Product-Focused Software Process Improvement. Springer, 2014, pp. 298–301.

[6] H. Terho, S. Suonsyrjä, A. Jaaksi, T. Mikkonen, R. Kazman, and H.-M. Chen, “Lean startup meets software product lines: Survival of the fittest or letting products bloom?” 2015.

[7] M. Meboldt, S. Matthiesen, Q. Lohmeyer et al., The dilemma of managing iterations in time-to-market development processes, 2012.

[8] Berente, Nicholas, and Kalle Lyytinen. "Iteration in systems analysis and design: Cognitive processes and representational artifacts." *Sprouts: Working Papers on Information Environments, Systems and Organizations* 5.4 (2005): 178-197, 2005.

[9] P. Kruchten, The rational unified process: an introduction. Addison-Wesley Professional, 2004.

[10] C. Sandor and G. Klinker, “A rapid prototyping software infrastructure for user interfaces in ubiquitous augmented reality,” Personal and Ubiquitous Computing, vol. 9, no. 3, pp. 169–185, 2005.

[11] C. Floyd, “A systematic look at prototyping,” in Approaches to prototyping. Springer, 1984, pp. 1–18.

[12] P. Pierre Bourque and R. e. Fairley, Guide to the Software Engineering Body of Knowledge, Version 3.0. IEEE, 2014.

[13] M. Fowler and J. Highsmith, “The agile manifesto,” Software Development, vol. 9, no. 8, pp. 28–35, 2001.

[14] S. Blank, The four steps to the epiphany. K&S Ranch, 2013.

[15] A. Maurya, Running lean: iterate from plan A to a plan that works. ” O’Reilly Media, Inc.”, 2012.

[16] M. McCurdy, C. Connors, G. Pyrzak, B. Kanefsky, and A. Vera, “Breaking the fidelity barrier: an examination of our current characterization of prototypes and an example of a mixed-fidelity success,” in Proceedings of the SIGCHI conference on Human Factors in computing systems. ACM, 2006, pp. 1233–1242.

[17] A. Taivalsaari and K. Systä, “Cloudberry: An html5 cloud phone platform for mobile devices,” Software, IEEE, vol. 29, no. 4, pp. 40–45, 2012.

[18] K. Schwaber and J. Sutherland, “The scrum guide,” Scrum Alliance, 2011.