Master's thesis

Master's Programme in Computer Science

Utilizing Software Analytics to Guide Software Development

Marko Juhani Koskinen

June 14, 2021

Faculty of Science

University of Helsinki

Supervisor(s)

Prof. T. Mikkonen, Ph.D. M. Luukkainen

Examiner(s)

Prof. T. Mikkonen, Ph.D. M. Luukkainen

Contact information

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki,Finland

Email address: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Faculty of Science	Master's Programme in Computer Science

Tekijä — Författare — Author

Marko Juhani Koskinen

Työn nimi — Arbetets titel — Title

Utilizing Software Analytics to Guide Software Development

Ohjaajat — Handledare — Supervisors

Prof. T. Mikkonen, Ph.D. M. Luukkainen

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's thesis	June 14, 2021	51 pages, 12 appendix pages

Tiivistelmä — Referat — Abstract

Modern software systems often produce vast amounts of software usage data. Previous work, however, has indicated that such data is often left unutilized. This leaves a gap for methods and practices that put the data to use.

The objective of this thesis is to determine and test concrete methods for utilizing software usage data and to learn what use cases and benefits can be achieved via such methods.

The study consists of two interconnected parts. Firstly, a semi-structured literature review is conducted to identify methods and use cases for software usage data. Secondly, a subset of the identified methods is experimented with by conducting a case study to determine how developers and managers experience the methods.

We found that there exists a wide range of methods for utilizing software usage data. Via these methods, a wide range of software development-related use cases can be fulfilled. However, in practice, apart from debugging purposes, software usage data is largely left unutilized. Furthermore, developers and managers share a positive attitude towards employing methods of utilizing software usage data.

In conclusion, software usage data has a lot of potential. Besides, developers and managers are interested in putting software usage data utilization methods to use. Furthermore, the information available via these methods is difficult to replace. In other words, methods for utilizing software usage data can provide irreplaceable information that is relevant and useful for both managers and developers. Therefore, practitioners should consider introducing methods for utilizing software usage data in their development practices.

ACM Computing Classification System (CCS)
Information systems applications → Decision support systems → Data analytics
Software and its engineering → Software creation and management → Software post-development issues → Software evolution

Avainsanat — Nyckelord — Keywords

software analytics, post-deployment data, software usage data

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Software study track

Contents

1 Introduction **1**

2 Background **3**

2.1 Software usage 3

2.2 Software usage data 5

2.3 Software analytics 6

2.4 Data-driven decision-making 8

3 Research approach **10**

3.1 Research questions 10

3.2 Structure of the study 10

3.3 Part I: Semi-structured literature review 11

3.3.1 Search String Definition 12

3.3.2 Inclusion and Exclusion Criteria 12

3.3.3 Data extraction 14

3.3.4 Concept Formation 14

3.4 Part II: Case study 14

3.4.1 Case selection 15

3.4.2 Case description 15

3.4.3 Data collection - Qualitative data from interviews 16

4 Semi-structured literature review **22**

4.1 Results 22

4.1.1 Manual analysis and simple visualizations 23

4.1.2 Visualizing software usage with heatmaps 24

4.1.3 Usage mining and pattern discovery 25

4.1.4 Constructing operational profiles 25

4.1.5 Machine learning and artificial intelligence 26

4.1.6 Aggregating usage data from several applications 26

4.1.7 Combining usage data with feedback and bug-reports 27
4.1.8 Data-driven design and experimentation 28
4.1.9 Data-driven requirements engineering 28
4.2 Validity 29

5 Case study **30**
5.1 Results 30
5.1.1 Defining software usage data and its uses 30
5.1.2 Usage of software usage data 31
5.1.3 Prototype evaluation 32
5.1.4 Putting the prototypes to real use 37
5.2 Validity 39

6 Discussion **40**
6.1 Analysis 40
6.1.1 Use cases of software usage data 40
6.1.2 Utilization methods of software usage data 41
6.1.3 Practicality of the prototypes 42
6.1.4 Success factors for software usage data utilization 43
6.2 Research questions revisited 43
6.3 Future work 45

7 Conclusions **46**

Bibliography **47**

A Sources of the semi-structured literature review

1 Introduction

Modern software systems are complex and the users of these systems have varying needs, competencies, and ways of making use of the software. According to the Pareto principle, also known as the 80-20 rule, 20% of the input produces 80% of the output [1]. By looking at this principle from the software development perspective, one might hypothesize that 20% of software system's features provide 80% of the value for the end-user.

The process of developing and maintaining modern software products creates a lot of data, including version control histories, build pass rates, feedback, bug reports, and real usage data. This data can be utilized in a variety of ways, for example, by combining it into visualizations to enable understanding the project's state at a glance [2, 3]. Often, however, even if there is a large amount of data available, it may be left unutilized.

To begin unleashing the power of the data, software practitioners have begun utilizing software analytics (SA). SA aims at increasing both the managers' and the developers' understanding of the software system by systematically condensing these large volumes of data into insightful and actionable metrics. Via these metrics, the development and maintenance efforts can be more efficiently targeted in a data-driven manner, instead of relying on intuition and speculation. Doing so is of high importance, since software development is an inherently expensive process. Misplacing the limited resources can lead to major repercussions from worsened customer satisfaction to project failure.

In this thesis, we investigate how SA can be integrated into the software development workflow of an agile software development team. The research approach is twofold. Firstly, we conduct a semi-structured literature review to learn about the methods and use cases for utilizing software usage data. Secondly, we conduct a case study to learn how developers and managers of an agile software development team utilize software usage data in their day-to-day work and to learn how they experience methods for utilizing software usage data.

This thesis is structured as follows. We begin in Chapter 2 by taking a look at relevant background literature. Next in Chapter 3, the research methodology to be used in this thesis is unfolded. In Chapter 4 the results of the semi-structured literature review are presented and their validity is evaluated. Continuing in Chapter 5, the results of the case study are presented, and their validity is examined. Next in Chapter 6, we discuss our

findings and their practical relevance. Finally, we conclude by summarizing our research efforts in Chapter 7.

2 Background

This thesis revolves around software usage and software analytics. This chapter presents some of the central concepts related to software usage and software analytics on a high level. We begin Section 2.1 by taking a look at how software systems are used in practice, and what are the contributing factors that affect the usage of a software system. In Section 2.2 we shed light on some of the common sources of software usage data. We continue in Section 2.3 by presenting an overview of software analytics. Finally, in Section 2.4, we conclude with an overview of data-driven decision-making.

2.1 Software usage

The usage of a software system is distributed among its features. The IEEE standard glossary defines a software feature as *"a software characteristic specified or implied by requirements documentation (for example, functionality, performance, attributes, or design constraints)"* [4]. In the context of this thesis, we mainly focus on functional requirements.

How the usage of a system is distributed among its features varies based on the user. Different users have varying needs, competencies, and ways of making use of the software. Past experiences with similar software systems may guide the user's behavior towards familiar ways of working, and the environment in which the software is used may also have an effect. Furthermore, the user's primary goal for using the software defines what parts of the software they will usually be interacting with. Even the user's personality traits may have an impact on how and when they use a particular software product [5]. Furthermore, the gender of the user has also been shown to have an impact on how the user perceives the usefulness and ease of use of the system, ultimately affecting if and how the user will be using the software [6].

Software usage is also highly dependent on the software itself. Social media software, such as Facebook, get quite even usage throughout the year. While, for example, the usage of software for filling in tax return forms is more tightly concentrated to specific times of the year. Sometimes these variations in the software usage patterns can cause major problems [7]. Naturally, the overall usability and the layout of the system's user interface also play an important role. For example, if accessing a feature requires the end-user to perform

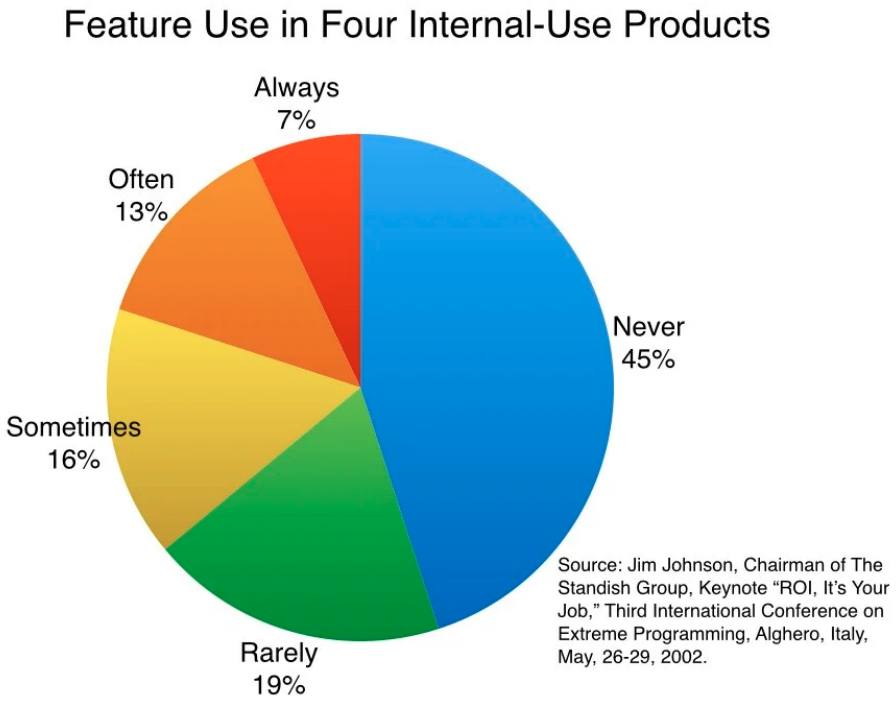

Figure 2.1: Feature Use in Four Internal-Use Products. From [9]. Original source [8].

a complex navigational process, the resulting usage of that particular feature might be lower than anticipated. Furthermore, when new features are added to the system, how are these additions communicated to the end-users that are supposed to benefit from them? If the users do not get informed or trained to use new features, the usage and therefore the value of these features might be less than optimal.

Some features of a software system are used more than others and some might never be used. A widely quoted study reports that 64% of four internal-use products' features are rarely or never used [8]. Figure 2.1 illustrates these results. Even though the aforementioned study has been widely quoted, the results have often been misinterpreted and overgeneralized [9]. Another single-case study reports that 28% of a software system's features were never used during a five-month period [10]. These two studies already illustrate that the usage of software systems is by no means homogeneous and highlight the importance of taking the data's time range into account when analyzing software feature usage. Generally speaking, there exists no consensus on what a typical software system's feature usage distribution might look like.

Even if some features of a system are never used, it does not necessarily mean that development efforts have been misplaced. It has been hypothesized that unused features may be justified if they make the consumer software product more attractive for potential customers [11]. Furthermore, a feature that implements a fail-safe mechanism for a nuclear reactor can be considered valuable, even though it was never to be used.

If a feature can not be justified, it probably should not be implemented. Unjustified features increase the development costs while providing no value. Besides, they unnecessarily increase the complexity of the system, making it harder to maintain and more prone to errors. As it happens, writing less code has been identified as an important strategy for making software development more productive [12].

Summing up, the usage of software systems is affected by a great number of factors. Thus, it is hard to accurately predict the actual usage of the system. Therefore to get an accurate representation of the usage of a system, one must turn to software usage data.

2.2 Software usage data

There exists a wide range of data sources that provide information on software usage, including web server logs and direct instrumentation of the software system. Both of which have unique benefits and drawbacks as a data source.

Web server logs store information about each request, for instance, the IP address of the client, what resource was requested, and so on. An important benefit of web-server logs is that they are automatically generated without any additional configuration. A drawback in this log-based approach is that due to the substantial caching practices of the modern web, the requests might never actually hit the server, decreasing the quality of the data [13].

Usage data can also be collected by purposefully instrumenting the software. Page tagging, for example, is a popular instrumentation method in the context of web applications [14]. It typically involves a snippet of Javascript code, which tracks the user's actions, such as what pages they visited, in what order, and how long they spent on those pages. Page tagging can provide more detailed information than the web server logs. It can, for example, report the browser's window size, and track custom events that do not automatically generate a request to a backend server. However, setting up page tagging requires some additional configuration, and the quality of data can be negatively affected by browser extensions

and security software that aim to prevent client-side instrumentation.

The problem with software usage data is that if it is not being utilized, it is just data. As it happens, many companies do collect huge volumes of post-deployment usage data, but only a few put it to use outside of diagnostic purposes [15].

No sensible manager or developer is willing to spend hours upon hours manually going through thousands of log file entries in search of new insights or innovations. Thus the raw software usage data needs to be processed further into insights via software analytics.

2.3 Software analytics

Analytics is the process of systematically analyzing data to support decision-making [16]. Software analytics (SA) aims to increase both the managers' and the developers' understanding of the software system by producing condensed information from raw data to empower rationalized decisions [17, 18]. SA plays an important role in modern software development [17]. In a similar vein, web analytics aims at achieving similar goals in the domain of web applications [19].

SA can be utilized to identify and address issues originating from any of the aspects of the trinity of software development, considering the software system, its users, or the software development process [20]. Figure 2.2 illustrates the trinity of software development. Identifying and addressing issues in the *software system* can enhance quality, for example, by locating bottlenecks in the application code or by identifying parts of the system which are potentially unsafe or unreliable. The *software users* perspective is more closely related to the real users' actions, expectations, and demands. Finally, the *software development process* perspective aims to improve the productivity of the development team, for example, by focusing development efforts on activities that have a high value/effort ratio.

SA can be used in different stages of the software development process. It can be used in the early stages, even before the system has been deployed to production. For example, in the implementation stage, software repository mining can already be applied on the source code to detect code smells early [21]. Once the development has been going on for some time, we can start answering questions such as: "How has the complexity and stability of our system evolved over time?", and "How should we target our testing and refactoring efforts?" [18]. Finally, when the software system gets deployed to production, more data becomes available. This makes it possible for SA to answer a wider range of questions,

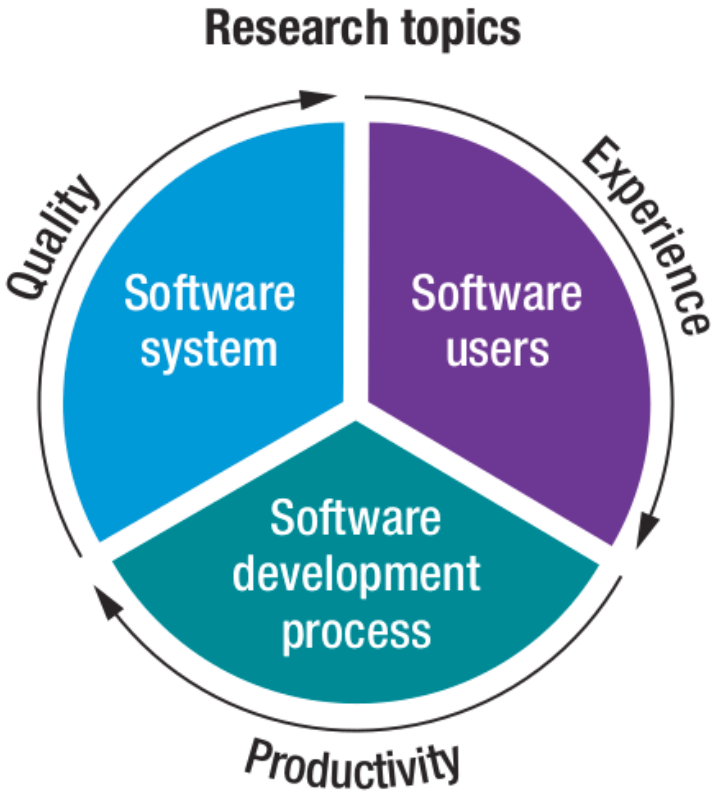

Figure 2.2: The trinity of software development. From [20].

such as: ”How are our users using the product?”, and ”How can we help and educate our users?” [18].

The results of SA should be both insightful and actionable [17, 20, 22]. Insightful results provide practitioners with useful information that can be put to use when working on a particular task. Actionable results consist of information that allows the practitioner to design and implement an (better, if an old solution exists) approach for dealing with the task at hand. A list of the software system's most commonly activated functions is an example of insightful and actionable results in the context of software development. This example is insightful because it communicates useful information to, say a project manager, who has to perform the task of backlog prioritization. It is also actionable because the project manager can realistically utilize this data when performing the prioritization process. The results must also be recent enough because outdated results are no longer actionable [17]. The required level of recency, however, is largely dependent on the appli-

cation domain. Furthermore, managers and developers are interested in different metrics. Thus the insightfulness and actionability of a particular SA metric depend on the person who is using it.

In practice, SA has proven to be both impactful and beneficial. Microsoft, for example, has demonstrated how performance data gathered from 48 000 users can be utilized to detect and resolve performance issues that would have otherwise remained undetected and unresolved [23]. Another company, Intuit, reports utilizing software's post-deployment data to facilitate product improvement and innovation [24].

2.4 Data-driven decision-making

As discussed above, the goal of SA is to empower rational decision-making by providing insightful and actionable metrics for developers and managers to use. As the name suggests, the main idea of data-driven decision-making is to make decisions based on data, instead of based on some stakeholder's opinions. The problem with basing decisions on the stakeholders' opinions is that opinions are subjective by nature and can turn the decision-making process into a politized one [24]. That is, the opinions of higher-ranked officials are often considered to be more important and correct, than those of their subordinates.

Data-driven decision-making is important because software development is an inherently expensive process. Misplacing the limited resources can lead to major repercussions from worsened customer satisfaction to project failure. Be that as it may, it has been shown that companies often do not actively even measure whether or not the features they are producing are of value to the end-users [25].

Fortunately, however, it has been shown that developers and managers alike are interested in utilizing various types of software metrics, based on real data, if made available [18]. Furthermore, it has been shown that both managers and developers see features as the most important type of artifact that should be analyzed [18].

Since developers and managers are working with features on a daily basis, it is no wonder that they are greatly interested in understanding feature-related artifacts. For example, a project manager might have to decide what features to prioritize in testing before the next release, while a developer might need to figure out how to resolve a bug report concerning a particular feature. In the context of this example, the manager might benefit from understanding how much existing features are being used to justify what features

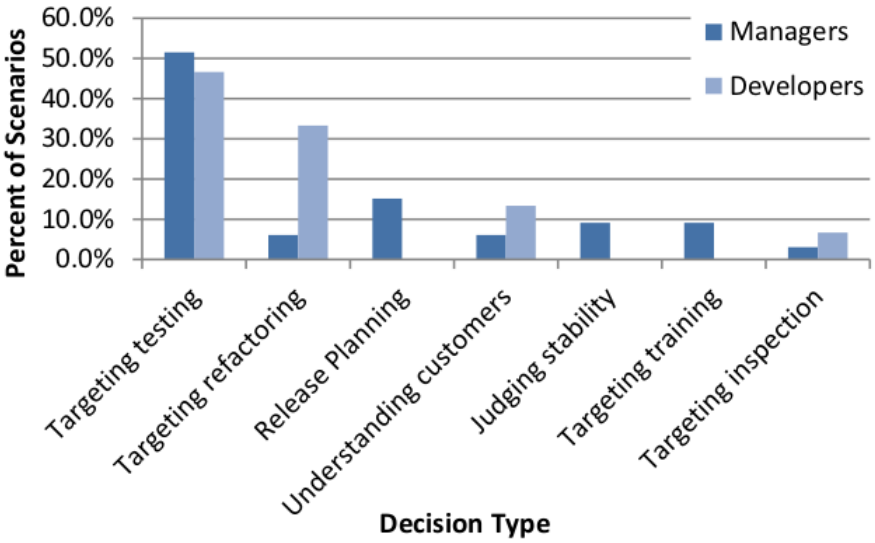

Figure 2.3: Types of decisions that analytics could help with. From [18].

should be tested. And the developer could benefit from understanding what exact chain of events caused the end-user to encountered the bug. Thus, providing the developers and the managers with feature-related metrics might be a good first step towards data-driven decision-making.

Software analytics is capable of enabling different types of data-driven decisions. In a study by Buse and Zimmermann [18], the information needs of software developers and managers were investigated. Figure 2.3 illustrates the types of decisions that SA could help with. Both parties are clearly most interested in metrics that could help with decisions related to software testing. Developers and managers are faced with different decisions. Thus, it makes sense that the developers and the managers are interested in using SA for different kinds of decisions.

3 Research approach

This chapter defines the research approach used in this thesis. We start by defining the research questions in Section 3.1, followed by a visual representation of the study design in Section 3.2. Then, in Section 3.3, the process for conducting the semi-structured literature review is presented. Finally, in Section 3.4, we bring forward the case organization and describe how the case study was conducted.

3.1 Research questions

In this thesis, we aim to enhance our understanding of the interplay between software usage and software development. We approach the problem via the following three research questions (RQs).

- RQ1: How and for what purposes can software usage data be used for?
- RQ2: How do developers and managers utilize software usage data in practice?
- RQ3: How do developers and managers experience methods for utilizing software usage data?

In RQ1, we investigate existing studies to learn how and for what purposes software usage data has been used in the past. In RQ2, we investigate how the developers and managers of an agile software development team utilize software usage data in their regular work by conducting a case study. As a part of the same case study, to answer RQ3, we evaluate the feasibility of software usage data utilization methods to learn how they would work in practice.

3.2 Structure of the study

The study consists of two interdependent parts: a semi-structured literature review, and a case study. Figure 3.1 illustrates the structure of the study. We begin by conducting a semi-structured literature review on past studies on utilizing software usage data to answer

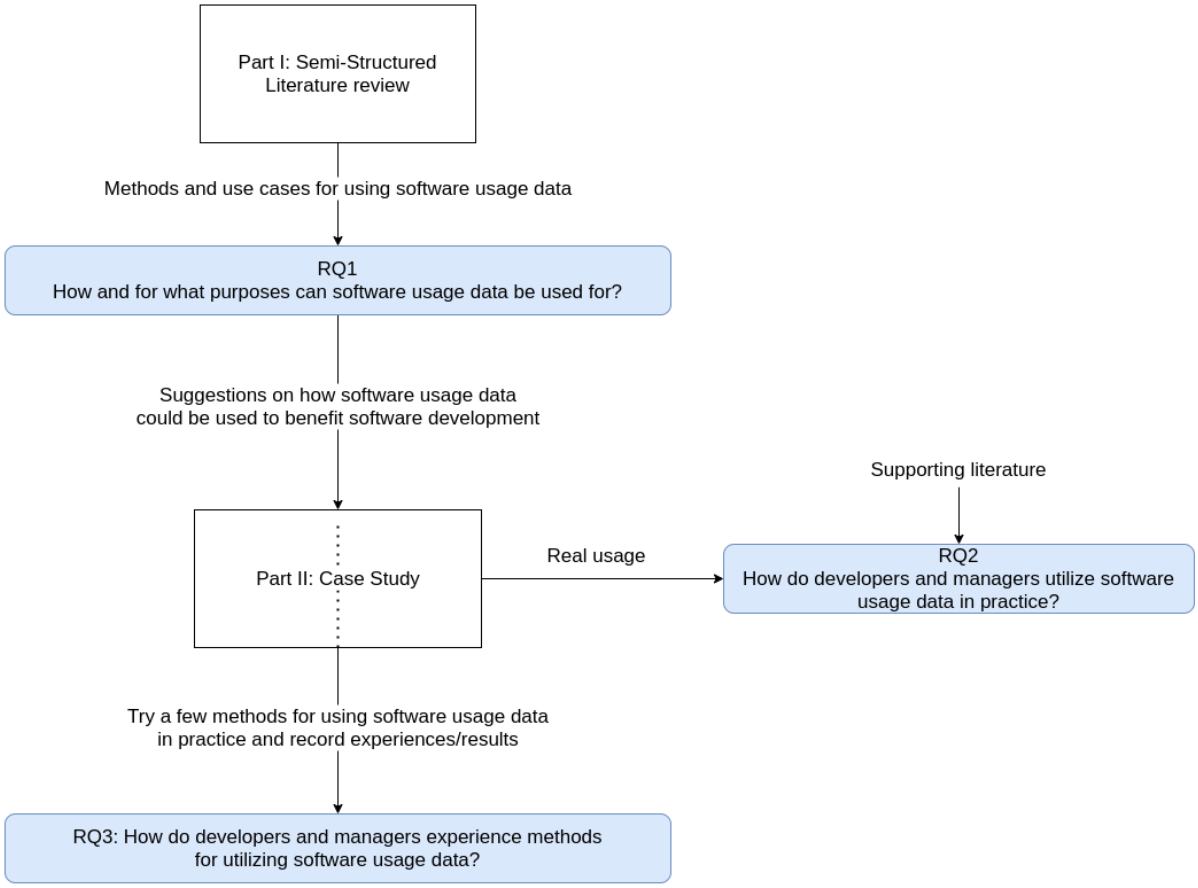

Figure 3.1: The study design.

RQ1. We continue with a case study to firstly determine how software usage data is used in practice, answering RQ2. And secondly, with past methods and use cases for software usage data charted out from the literature, we have a solid foundation for experimenting with a subset of the methods in a form of a case study to answer RQ3.

3.3 Part I: Semi-structured literature review

This section describes how the semi-structured literature review (SSLR) is conducted. The goal of the SSLR is to find answers to RQ1: *How and for what purposes can software usage data be used for?* In other words, the goal is to discover methods for utilizing software usage data and to learn what use cases those methods can help us achieve.

3.3.1 Search String Definition

The search was performed on the abstract and citation database Scopus with the following search string on the 1st of February 2021, resulting in 99 preliminary results. All the 99 sources are listed in detail in Appendix A.

```
(
    TITLE-ABS-KEY("usage mining" OR "usage data")
    AND TITLE-ABS("software")
    AND TITLE("experience" OR "report" OR "case study")
    OR ABS ("experience" OR "report" OR "case study")
)
```

3.3.2 Inclusion and Exclusion Criteria

After the initial search resulting in 99 hits, abstracts were read and evaluated against the inclusion/exclusion criteria. In a few unclear cases, some sources were read in more depth to make a justified decision. Three of the sources discussed the same method, REQANALYTICS. Only the most recent source (S19) discussing the method was included.

The source must meet all of the following inclusion criteria:

- The title or the abstract must strongly hint that software usage data is used in the study.
- The study must propose or experiment with a method of using software usage data. Or the study must describe a use case for software usage data.

The source must not meet any of the following exclusion criteria:

- The source focuses on a software system that collects or processes other than software usage data.
- The source is a workshop introduction.
- The source is a listing of a proceeding's contents.

After applying the inclusion/exclusion criteria, 49 sources remained for further analysis. Table 3.1 presents the included sources' titles along with an assigned SID (source identifier) that will be used to refer to a particular source later in this thesis.

Table 3.1: Included sources for data extraction and concept formation.

SID	Title
S1	Automation of NERSC Application Usage Report
S2	Can operational profile coverage explain post-release bug detection?
S3	Prediction of Success and Complex Event Processing in E-Learning
S4	Enabling Data-Driven API Design with Community Usage Data: A Need-Finding S...
S5	Lessons learned from developing and evaluating an educational database mode...
S6	Quantifying use of a health virtual community of practice for general pract...
S7	How do practitioners capture and utilize user feedback during continuous so...
S8	Don't worry, be happy - Exploring users' emotions during app usage for requ...
S9	Development of a web based framework to objectively compare and evaluate so...
S10	FAME: Supporting continuous requirements elicitation by combining user feed...
S11	Physician experience with speech recognition software in psychiatry: Usage ...
S12	A machine learning approach to generate test oracles
S13	Feature crumbs: Adapting usage monitoring to continuous software engineerin...
S14	Notifying and Involving Users in Experimentation: Ethical Perceptions of So...
S15	Behavior metrics for prioritizing investigations of exceptions
S16	ForgetMeNot: Active reminder entry support for adults with acquired brain i...
S17	Application of adaptive game-based learning in image interpretation
S18	Exploring use and benefit of corporate social software: Measuring success i...
S19	Maintaining Requirements Using Web Usage Data
S20	Toward data-driven requirements engineering
S21	Improving web navigation usability by comparing actual and anticipated usag...
S22	Early experiences in developing and managing the neuroscience gateway
S23	UX work in startups: Current practices and future needs
S24	Vision 2020: The future of software quality management and impacts on globa...
S25	Usage results of a mobile app for managing urinary incontinence
S26	Strengthening district-based health reporting through the district health m...
S27	Deploying communitycommands: A software command recommender system case stu...
S28	Experiences gamifying developer adoption of practices and tools
S29	Fixing the 'out of sight out of mind' problem: One year of mood-based micro...
S30	Patina: Dynamic heatmaps for visualizing application usage
S31	Data-Driven Design Process in Adoption of Marking Menus for Large Scale Sof...
S32	Post-deployment data collection in software-intensive embedded products
S33	Development and feasibility of an electronic white blood cell identificatio...
S34	A case study on usage of a software process management tool in China
S35	Integrating quality, quality in use, actual usability and user experience
S36	Model-driven instrumentation of graphical user interfaces
S37	Interactive usability instrumentation
S38	Web usage mining in a blended learning context: A case study
S39	Seeking activity: On the trail of users in open and community source framew...
S40	Value tensions in design: The value sensitive design, development, and appr...
S41	Evaluation of the Turkish translation of the Minimal Standard Terminology f...
S42	Improving web service discovery with usage data
S43	Building an institutional repository: Sharing experiences at the HKUST libr...
S44	Intelligent decision support system based on data mining: Foreign trading c...
S45	Analysis of user's behaviour in very large business application systems wit...
S46	Designing adaptive mobile applications: Abstract components and composite b...
S47	A provision framework and data logging tool to aid the prescription of elec...
S48	Experiences with an interactive video code inspection laboratory
S49	Remedial and second language English teaching using computer assisted learn...

3.3.3 Data extraction

After applying the inclusion/exclusion criteria, the following properties were extracted from each of the remaining 49 sources.

- SID as defined in Table 3.1.
- Title and abstract of the paper.
- A collection of relevant quotes.
- A description of the method(s) used for utilizing software usage data, if presented.

The data were collected in a separate document for concept formation purposes.

3.3.4 Concept Formation

Two distinct types of concepts were formed 1) use cases for software usage data, 2) methods for utilizing software usage data. The distinction is important. While all of the included sources discuss use cases for software usage data, not all of them describe concrete methods for achieving those use cases.

From the 49 included sources, the extracted quotes were assigned to use cases for software usage data by employing the concept-centric approach [26]. Initially, 26 distinct use cases for software usage data were identified, 15 of which were present only in two or fewer sources. A closer look at these rare concepts was taken to merge them into larger ones when appropriate.

As for forming the concepts for methods of utilizing software usage data, the same extracted data from the same 49 included sources were used. In particular, the extracted descriptions of the method(s) were used to group similar methods under the same concept.

3.4 Part II: Case study

As illustrated in Figure 3.1, case study acts as an information source for addressing both research questions two and three. We start by exploring how a small and agile development team utilizes software usage data in their work, allowing us to answer RQ2: *How do developers and managers utilize software usage data in practice?* Then, to answer RQ3:

How do developers and managers experience methods for utilizing software usage data?, we analyze the feasibility of two previously identified methods for utilizing software usage data by constructing and evaluating their prototypes with three developers and a consultant of the development team.

3.4.1 Case selection

The case organization was selected by convenience. The author has previously worked in the organization as a software developer. The manager of the organization had also previously indicated to be interested in further investigating how the unutilized potential of software usage data could be unleashed. The case application was selected because it is the most information-rich one available in the organization and it has already been gathering feature-based usage data for over a year.

3.4.2 Case description

The context of this study is a software development team, working on a certain computer application. Table 3.2 summarizes the composition of the development team. Each member of the team has been assigned an identifier that will be used when needed to refer to a particular member. The development team itself consists of a project manager (I1), a consultant (I2), and three full stack developers (I3-I5). The consultant (I2) acts as a middle-man between the end-users and the development team.

Table 3.2: Compositions of the case application's development team.

Identifier	Role	Experience with the case application
I1	Project manager	three years
I2	Consultant	three years
I3	Full stack developer	three months
I4	Full stack developer	over a year
I5	Full stack developer	a year

The team values both agile and lean principles by aiming to satisfy the end-user's needs while eliminating waste. At the time of the study, the team was working completely remotely. The development takes place in short iterations, creating a need for continuous monitoring of the system's state and learning from software analytics. Besides, the team

works in a self-organizing manner, which builds up an increased need for some metrics to guide their work. As pointed out by Martínez-Fernández et al. [27], software analytics can be especially useful for teams practicing agile software development.

The case application is a tool for visualizing and presenting large amounts of complex data to support generating insights and decision-making. The users of the application are teaching and management personnel.

3.4.3 Data collection - Qualitative data from interviews

In the interviews, we collect qualitative data to be used for answering both research questions two and three. We begin the interviews by exploring the topic of RQ2: *How do developers and managers utilize software usage data in practice?* Then, we explore the topic of RQ3: *How do developers and managers experience methods for utilizing software usage data?*, by evaluating two prototypes of software usage data utilization methods with four members of the case application's development team.

A longitudinal study examining the actual effects of putting the methods to use was not possible because 1) the timeline of the thesis did not allow it, 2) the case organization's resources had to be focused on more time-critical activities at the time of the study.

Selected methods and their prototypes

Two methods for utilizing software usage data were selected from the findings of the semi-structured literature review, presented in Table 4.1 on page 22. The two methods were selected by the author based on his knowledge of the available software usage data, the project manager's needs/wishes, and the author's personal interest. Not all methods were feasible in the context of this case study to begin with, for example building an AI system for processing usage data is clearly out of the scope of this thesis. The two selected methods were: *Combining usage data with feedback and bug-reports* and *Visualizing software usage with heatmaps*. The prototypes are displayed in Table 3.3 and Figure 3.2 respectively.

Combining usage data with feedback and bug reports, or more specifically combining usage data with bug reports was selected because the case application has both usage data and automatically collected bug reports available, making the method a feasible one. Furthermore, the project manager had previously indicated that they would like to see if and how these two sources of data could be used together. Before reaching its

Table 3.3: Prototype 1: Combining usage data with bug reports. The three-digit codes in the Errors-column stand for standard HTTP response status codes [29].

URL	Errors	Pageviews	Total users	**Users affected**	**Error rate**
/coursestatistics	16 x 500 17 x 400 8 x 502 1 x 503 42 total	325	233	7 (3% of users)	13%
/students	38 x 400 2 x 401 40 total	850	200	4 (2% of users)	5%
/populations	8 x 401 4 x 400 2 x 403 2 x 503 1 x 502 17 total	323	243	6 (2.5% of users)	2%

final form, the prototype was enhanced based on feedback from senior researchers. The prototype is presented in Table 3.3. It was constructed by combining usage data from web analytics software and automatically generated bug-report data from specialized exception capture software Sentry [28]. Sentry captures any exceptions thrown by the application and is capable of determining how many users have been affected by the same exception. However, it does not monitor the normal usage of the system. Thus, to be able to calculate the error rate and how many users are affected by the errors, usage data has to be obtained from a different source.

Combining the data was straightforward since Sentry captures the URL in which the error occurred, while web analytics reports page views based on the URL. After combining the data, the error rate of each particular URL was calculated with the following formula: *errorRate = errorsTotal/pageviews*. To keep the prototype actionable, the time range of data for this prototype was set to be from the 1st of February to the 1st of April 2021. Only the three most error-prone URLs were included.

Visualizing software usage with heatmaps was selected as the second method. The method is feasible, because the case application's usage data includes interface-level events, making

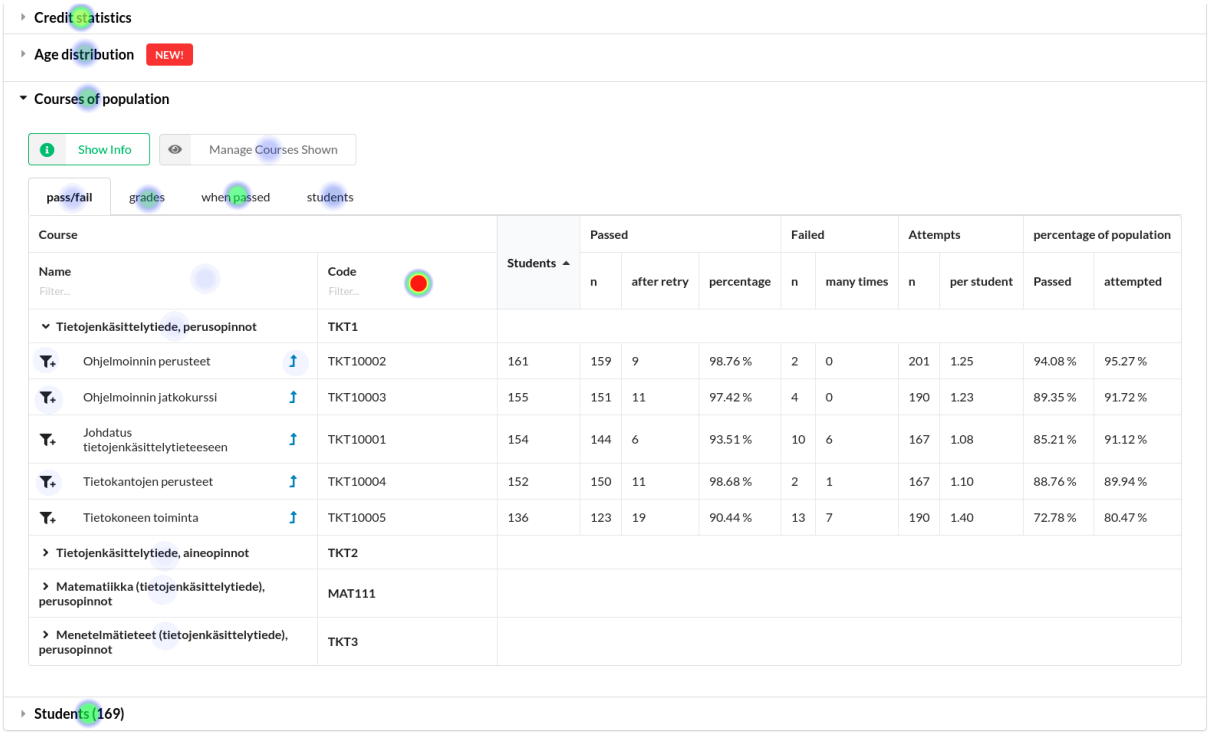

Figure 3.2: Prototype 2: Visualizing software usage with heatmaps.

the creation of a heatmap overlay possible. The heatmap overlay was created by combining a screenshot of the graphical user interface with a heatmap generated with heatmap.js [30]. After deciding on the view to be visualized, usage data for the interface level elements were gathered manually. The process of mapping the collected usage data to the interface elements required a moderate amount of manual work.

Figure 3.2 presents the prototype of the heatmap overlay. The used data range was from 2020 to the 1st of April 2021. The colored dots indicate the relative usage of each particular user interface element. For example, we can see that the *Name* filter is being used significantly less when compared to the *Code* filter. This particular portion of the application's user interface was selected for the prototype because it has been thoroughly instrumented and illustrates a heterogeneous usage distribution.

Interview questions

A semi-structured interview was selected as the type of the interviews as it is a suitable format for gathering descriptive and explanatory data while retaining flexibility [31]. The interview questions were designed to contain both closed and open-ended questions. Before reaching their final form, the interview questions were iterated on based on feedback from senior researchers. The final set of interview questions is as follows:

1. How would you describe your role in the case application?
2. How long have you been working with the case application?
3. How would you define the term ”software usage data”?
    - 3.1. What types of data does it include/exclude?
    - 3.2. Hypothetically, how and for what purposes (goals) could the data be used for?
4. What do you think about the idea of putting software usage data to use?
5. How is software usage data being utilized currently in the organization?
    - 5.1. What benefits have been achieved?
    - 5.2. Are these some difficulties related to using the data?
    - 5.3. Have you personally used the project's usage data for something before?
        - 5.3.1. For what?
        - 5.3.2. How often?

At this point, the prototypes are presented.

6. Please rate the prototypes, on the scale of 1–5, on the following dimensions **according to your own experience**:
    - 6.1. Understandability
    - 6.2. Relevancy
    - 6.3. Informational value
    - 6.4. Irreplaceability
7. For ratings of 1–2 and 4–5 can you elaborate why so low or high rating was given?

8. What did you learn from the prototype?
    8.1. In what situations could the learned information be most useful in?
    8.2. What kinds of decisions could the learned information support?
9. What benefits and drawbacks does the prototype have?
10. How would you improve the prototype?
11. How would you feel about putting the prototype to actual use?

For the remaining questions, the interviewees were asked to imagine putting the proposed prototypes to real use in the case application

13. What high-level tasks would have to be performed and by whom?
14. How much time and resources would be required?
15. How often would it make sense to make update/reproduce the prototypes?
16. Is there something that might hinder putting the prototypes to use?
    16.1. Do some other tasks take priority?
17. Is there something that might help with putting the prototypes to use?
18. Is there anything you would like to add or a particular topic to revisit?

Conducting the interviews

The interviews were conducted one-on-one with four members of the development team, including the consultant (I2), and the three full stack developers (I3, I4, I5), as described in Table 3.2. The project manager (I1) was not included in the interviews, as their active involvement in the thesis work would likely have had an impact on that interview's results. Even though the three developers share the title of *"full stack developer"*, they have been working on different aspects of the application, ranging from design and frontend development to DevOps-related activities. Members with different roles were included to capture data from multiple different perspectives [31]. The interviews were conducted remotely and in Finnish. Audio recordings were collected from each of the interviews with each individual's consent for transcription purposes.

In each interview session, the author presented the prototypes to the interviewee. In the brief presentation, the author described how the prototypes were constructed, what data were used, and how much effort was required. The interviewees were encouraged to ask any questions they might have regarding the prototypes during and after the short five-minute-long presentation. The prototypes were purposefully presented after covering the first five interview questions so that the results of those questions would not be affected by the presentation. The interviewees were given access to screenshots of the prototypes to help them answer questions regarding the prototypes.

Data analysis

After conducting the interviews, raw data in the form of five audio files were transcribed by the author. The transcribed interviews were then dissected via thematic analysis [32]. Once the data had been processed to its final form, the results were brought back to the interviewees for member checking to catch any misunderstandings [33]. No issues were identified.

4 Semi-structured literature review

This chapter presents the results of the semi-structured literature review. Furthermore, the validity of the results is evaluated. We start in Section 4.1 by presenting the findings of the SSLR. After presenting our findings, we conclude in Section 4.2 by examining the validity of the results.

4.1 Results

A total of 9 methods and 9 use cases for software usage data were identified. Table 4.1 presents the identified methods in a descending order based on the number of sources discussing each particular method. Table 4.2 presents the identified use cases in a similar manner.

Table 4.1: Identified methods for utilizing software usage data.

Method	Count
Manual analysis and simple visualizations	20
Usage mining and pattern discovery	4
Data-driven requirements engineering	2
Machine learning and artificial intelligence	2
Aggregating usage data from several applications	2
Combining usage data with feedback and bug-reports	2
Constructing operational profiles	1
Data-driven design and experimentation	1
Visualizing software usage with heatmaps	1

In the following, the identified methods and their respective use cases for utilizing software usage data will be presented to answer RQ1: *How and for what purposes can software usage data be used for?* Use cases will be connected to each method to describe what are the potential benefits that the method could bring. A single method can be used to achieve several use cases. Furthermore, many use cases can be achieved via multiple methods. For presentation purposes, some closely related methods have been grouped together.

Table 4.2: Identified use cases for software usage data.

Use case	Count
To better understand software usage	27
To locate issues and improvement needs	17
To enable measuring, monitoring and evaluating software	13
To understand the users' behaviors and needs	10
To enable comparing different applications and different versions of the same application.	4
To enable making rationalized design and development decisions	4
To help with support and bugfixing activities	3
To help with software testing	3
To enable reporting of usage data to external stakeholders	2

4.1.1 Manual analysis and simple visualizations

The most common method for using software usage data is to perform manual analysis on the data and visualize the results. Manual analysis of software usage data has plenty of potential use cases. Most importantly, it can help stakeholders better understand how the software system is truly being used on a variety of dimensions. Depending on the type and level of detail of the usage data, the dimensions can range from estimating the number of active users to minute-scale breakdowns of the system's feature-level usage. If the data allows, the usage of a feature can further be tied to a specific version of the feature, enabling the comparison of feature increments (S13).

Visualizing the results of manual analysis via simple methods, such as clustering and plotting, can also act as a powerful way to grasp a better understanding of software usage. Furthermore, such visualizations can also act as a rudimentary method for finding patterns and trends in the data.

Both of the techniques can also allow for a better understanding of the users' behaviors and needs. These techniques can also help to detect issues in the software or its use. S3 for example was able to identify a problematic user behavior by manually analyzing educational software's usage data:

> *The students use this "check button" a bit more than we expected, especially for difficult and large exercises. [...] One explanation for this is that even though the*

> *check button in LearnER shows explanations for all errors in the model, students often fix only one or two errors, and then click the check button again.* (S3)

Manual analysis and simple visualizations can also be used to evaluate whether or not a software is fit for its purpose. Studies S16, S25, and S47, for example, utilize these techniques to evaluate the fit of medical software systems. S16 performs statistical analysis to determine whether or not unsolicited prompts can be of benefit for people with memory problems related to acute brain injury. S25 utilizes usage data to measure the acceptance of a mobile application by calculating the number of users who only use the application once, and to evaluate the hypothesis that the application can help manage a particular medical condition. S47 visualizes usage data with bar- and line charts to discover answers to questions such as 'Is the interface most efficient for the user?', and 'Does the application usage follow the expected pattern?'.

4.1.2 Visualizing software usage with heatmaps

If usage data can be linked to interface elements, as is often the case, a new visualization technique can be unlocked.

S30 presents Patina, a system for gathering and visualizing software usage data by applying a heatmap on top of the application's user interface. In addition to being able to visualize the user's own usage data, the system also allows the user to see what the heatmap of the community (other users) looks like. The system does not extend beyond desktop applications, but the idea of visualizing usage data with heatmaps remains a possibility.

The study claims that visualizing an application's usage data by applying a heatmap overlay has three main potential benefits for the system's users. Firstly, it can help new users to get to know the system by seeing what interface elements a typical user interacts with. Secondly, if the usage data can be tied to a specific document or more generally to any artifact that can be shared and manipulated by a group of users, the most commonly used interface elements for that specific document/artifact can be highlighted. Thirdly, the user can potentially learn from the way other users use the software and also reflect on their usage on the wider population.

> *For example, one user discovered the "zoom slider" in the bottom right corner of the window, and has subsequently adopted the use of that slider for zooming his documents.* (S30)

4.1.3 Usage mining and pattern discovery

Another common method for utilizing software usage data is pattern discovery through usage mining. Unlike manual analysis and visualization techniques which cover a wide range of use cases, pattern discovery has more limited applications. The main focus of pattern discovery in the context of software systems lies in detecting problematic patterns that require attention.

S21, for example, uses web usage mining techniques on software usage data to discover the actual usage patterns, which are then compared with the anticipated usage patterns constructed by experts in human cognition. Detecting differences in the patterns can be indicative of poor design choices in the software that should be addressed to enhance the usability of the system:

> *Temporal deviations also highlight some usability problems linked to pages where users spent excessive time. For example, Table II shows that 23 users spent excessive time in the page "register.php." After inspecting this page, domain experts recommended that some page elements be redesigned to enable more efficient completion of this task...* (S21)

According to S4, there is also an interest in finding re-occurring patterns from usage data, in addition to finding differences in expected and actual usage patterns. Such patterns can be indicative of users using the software incorrectly. The same study, however, points out that the tool support for discovering such patterns is lacking.

4.1.4 Constructing operational profiles

An operational profile defines how the usage of a system is expected to be distributed over its features and functionalities. Operational profiles are essential in operational testing, where the goal is to prioritize limited testing resources on the most crucial parts of the software system. By targeting the limited testing resources on the most crucial parts of the system, the highest possible level of reliability can be achieved.

S2 describes how an operational profile can be constructed from software usage data. The resulting operational profile estimates, on code-level, the probabilities of each function being called. The authors show, that by further combining the operational profile with test coverage data, the quality of the projects testing efforts can be better evaluated.

Usage data is not required for creating an operational profile. Instead of usage data, other

data sources such as user interviews can be used. Furthermore, oftentimes usage data is not available since testing usually takes place before shipping software to production.

4.1.5 Machine learning and artificial intelligence

Software usage data can be utilized as an input source for machine learning (ML) and artificial intelligence (AI) systems. S3 describes a ML system that both collects and processes usage data from log files of an educational software system. The goal of the system is to generate actions that might be able to benefit the user's learning process. S17 presents a similar system, designed for adapting a game's difficulty based on the user's performance.

S12 presents a supervised ML approach for generating test oracles based on real usage data. The test oracle infers/learns the correct behavior of a software system based on the real usage of that system. For example, if a user performs the *action* of logging in to the system, the oracle learns that the system *should* show a welcome-message to the user. Now, if a programmer were to accidentally break the login functionality, the same action of logging in would no longer result in the system showing a welcome-message. Thus a difference between the anticipated and actual outcomes would be detected, indicating that there may be a bug in the system. On a high level, the approach is quite similar to S21's approach of comparing actual and anticipated usage patterns.

> *The main idea is to compare the captured information of the application with the oracle information referring to the same action, in order to identify any inconsistencies in the application behavior.* (S12)

While the examples presented here may not be easily transferrable to other contexts, they do illustrate a range of possibilities for software usage data in the domain of AI and ML systems.

4.1.6 Aggregating usage data from several applications

Many methods for utilizing software usage data deal with systems that aggregate usage data from multiple different applications. Doing so unlocks some new use cases for software usage data.

S1, for example, presents a system that collects usage data from thousands of different jobs running on the same scientific computing hardware in the same physical location.

Gathering the usage data to such a central system has several benefits. Firstly, it enables comparing the usage of different applications. Secondly, it can make the task of reporting application usage to external stakeholders significantly easier by decreasing the amount of manual work required.

If aggregated to a central location, usage data from otherwise completely separate applications can also be used as a basis for recommender systems. S42 presents an approach for collecting and processing usage data from service-oriented applications to generate recommendations of web services for software developers to use. The proposed system is fundamentally different from most other recommender systems in that it relies on implicit feedback (usage data) instead of explicit feedback, such as ratings and reviews.

Usage data can also be collected systematically and purposefully to enable the comparison of software systems. S9 presents a framework for comparing any two web-applications to detect issues related to usability and user experience and to objectively evaluate which of the two performs better.

4.1.7 Combining usage data with feedback and bug-reports

Software systems get feedback through a variety of channels including emails, phone calls, and special feedback-functionalities integrated within the software system. Studies suggest that usage data could and should be taken into account when handling feedback. It has also been claimed that usage data itself can be used as implicit feedback, even if there is no explicit feedback available.

S10 presents a system for combining user feedback with software usage data. The authors argue that if feedback can be combined with usage data of the system, the end-users' needs can be better understood. A case study is presented, highlighting several use cases where the approach can be beneficial.

Firstly, if the feedback concerns a functionality of the system, usage data can be taken into account in the prioritization process. For example, if a user reports a bug in a feature, usage data can be taken into account to estimate how many users the bug is affecting. S15 goes more in-depth in estimating user-inconvenience by investigating the actions taken by the user after encountering a bug. By understanding what happens after a user encounters a bug, the bug's severity can be better evaluated. For example, if the bug causes the whole application to crash multiple times, the bug is likely of high priority. But if the bug only causes the user to repeat the same action, a lower priority rating is probably justified. The

case study presented in S15 suggests that if bug-reports are combined with estimations of user-inconvenience, the prioritization of roughly every 3rd bug-report is likely to change.

Secondly, usage data can help understand the context in which the feedback was given and thus the meaning of the feedback. S10 presents an example, where a user sent feedback about a problem they encountered, but a closer inspection of the user's usage data revealed that the actual problem was different than the reported one.

4.1.8 Data-driven design and experimentation

Data-driven design practices rely on data to make educated design decisions. S31 describes the success story of Autodesk in designing and implementing a new marking menu for their software system.

> *We analyzed the usage data from the past release to identify high usage commands and coverage of those commands. It showed that top 20-30 commands in the workspace of Inventor takes 80% of workflow. This result supported the single level of menu instead of hierarchical menu, while identifying the commands that we need to put in.* (S31)

Software usage data is essential for development practices that rely on experimentation. One such development practice is A/B testing, where a single variable is altered and the effect of the change is evaluated. A simple experiment for an A/B-test would be to evaluate whether a red or a green button get more clicks. The data that is evaluated in A/B testing, and other types of experimentation, can originate from many sources, including the real usage data of a software system.

4.1.9 Data-driven requirements engineering

Software usage data can also be put to use while performing the process of requirements engineering.

S20 hypothesizes that the trend in requirements engineering might be towards a more data-driven process, where various sorts of data stemming from the end-users will be playing an increasingly important role. However, the authors also acknowledge that there are still plenty of challenges and issues ahead on employing data-driven requirements engineering on a large scale.

> *We project a transition from stakeholder-centered, intuition- or rationale-based decisions to group-based, mass-driven, user-centered decisions. Such decisions will be based on real-time analysis of a range of information sources.* (S20)

S19 presents a system called REQANALYTICS, which aims to help with the requirements management process by generating recommendations for actions. The system functions by combining manually specified functional requirements with web usage data. It is capable of generating several types of recommendations, including creating and splitting requirements and adjusting priorities.

Furthermore, when considering an end-user's request for a particular enhancement, usage data can be of help. S10 discusses an example, where an end-user wanted to know how a "conversion factor" should be calculated. The stakeholders' first impression was that the application should of course calculate this conversion factor automatically. However, after consulting the usage data it was determined that such an enhancement was not justified due to anticipated low usage and that the average user was able to compute the required conversion factor quickly. Thus it was decided that adding instructions for calculating the conversion factor would suffice.

4.2 Validity

The search string may have not been optimal, resulting in a biased population of the discovered methods and use cases. Most importantly, it was noted that only very few papers were discussing continuous experimentation as a method for utilizing software usage data. Furthermore, the frequencies of the methods and use cases should be taken with a grain of salt. Altering the keywords would likely result in different frequencies.

There exists no categorization for the methods and use cases for software usage data. Thus, the methods and use cases defined here by the author, may not be the same definitions as other researchers would have ended up with.

5 Case study

This chapter presents the results of the case study. Furthermore, the validity of the results is evaluated. We begin in Section 5.1 by presenting the interview results and conclude in Section 5.2 by examining the validity of the results.

5.1 Results

This section presents the interview results. We approach the presentation by dividing the results into four major categories 1) defining software usage data and its uses, 2) usage of software usage data, 3) prototype evaluation, and 4) putting the methods to real use.

5.1.1 Defining software usage data and its uses

The interviewees gave homogeneous definitions for software usage data. All of them described that software usage data conveys how the users *actually* use the software. Examples of data were given, such as 1) what buttons the users click in the application, 2) how many users the application has overall, and 3) how much time the users spend in the application.

As for the sources of software usage data, there were some differences. Two out of four interviewees mentioned application logs as a source of software usage data, while the rest only mentioned client-side tracking techniques such as Google Analytics.

When asked what hypothetical goals could be achieved by utilizing software usage data, the interviewees reported an overlapping collection of activities. The developers thought that software usage data could be used for many software development-related tasks, including 1) prioritizing and planning feature development, 2) assisting in debugging, and 3) helping to discover underused features. As an example, one of the developers described a usage scenario, where software usage data could be of value to help to target development efforts and to identify issues:

> *The project manager or someone else might look at the data and say that maybe this feature is not so important as no one is using it. Therefore we should focus on the more important features. Or they might notice that some feature is not being*

> *used and then begin to think why that is. From the developers' perspective: is there something wrong with the product?* (I4)

The consultant was also interested in discovering underused features, but also in understanding what groups of users are using the product. As described by the consult knowing what groups of people make use of the system is important for their work.

The overall attitude towards putting software usage data to use was positive. One developer described that if software usage data is not available, then to get feedback one must resort to asking the users for it. This approach was seen as being too laborious. Besides, it was hypothesized that the users would be more inclined to leave negative feedback as opposed to positive feedback, which was seen as an unpleasant outcome. It was also emphasized that when compared to other data collection methods, collecting software usage data is painless because once it has been set up, the collection process is automatic.

Some negative aspects were identified as well. One developer noted, that while the usage data has some value, it is more important to have direct communication with the end-users:

> *It's a double-edged sword. It (software usage data) can be a great source of information because data always tells about something. But one should not focus on the data too much. I think that talking with the end-users or seeing how they use the product is more important and that the data should play a more supportive role.* (I3)

Furthermore, another developer expressed their concern about the added infrastructural complexity that comes with implementing the collection and storage of the usage data. It was also mentioned that the granularity of automatically collected data can not match the detail of other feedback collection methods, such as user interviews.

5.1.2 Usage of software usage data

How software usage data is and had been used varied greatly among the interviewees. Some reported utilizing it almost weekly, while some reported needing it only very seldom or not at all. Two of the developers mentioned debugging as the most common use case for the usage data. In the process of debugging complex problems that were hard to reproduce, the developers turned to usage data to learn what were the steps that the user took that resulted in the error.

Furthermore, all of the developers mentioned a recent refactoring effort of a particular part

of the application that was driven forth by usage data. This refactoring process took wind when the project manager noticed, based on usage data, that a part of the application was being underused. The developers then took a closer look at the data to determine what features should be kept and which should be eliminated from the new version, based on the usage data of said features:

> *I recall that we used the data to make some decisions regarding refactoring the filters. By manually just looking at the data we were able to determine which of the filters should be removed from the new version.* (I4)

The consultant mentioned preferring to ask any usage data-related questions from the developers or the project manager, instead of looking at the data themselves. For the consultant, the data had been used indirectly to detect underused features. After detecting such features the consultant along with the developers approached the end-users of the application to try to determine why the features were not being used.

The developers had experienced some difficulties while putting the data to use. One developer mentioned that interpreting the data can be difficult because there is no baseline for what the usage of a successful or an unsuccessful feature looks like. Furthermore, a developer noted that depending on the quality of the data collection implementation, the data might be difficult to interpret and understand, or downright faulty:

> *If the data collection has been poorly implemented, then it can be difficult to understand and make use of the data. [...] If a user has some URL containing a query in their bookmarks, then the buttons required to make that query don't get pressed. So even though the query is being made, it might not show up in the data as one would expect.* (I4)

5.1.3 Prototype evaluation

For presentation purposes, prototypes 1 and 2 will hereon out be referred to as P1 and P2, respectively. Numerical results of the prototype evaluation are presented for P1 and P2 in Figure 5.1 and Figure 5.2, respectively. Both of the prototypes received relatively good overall scoring. P1 achieved a total score of 61/80 and P2 60/80. Next, we take a look at each of the dimensions for each of the prototypes.

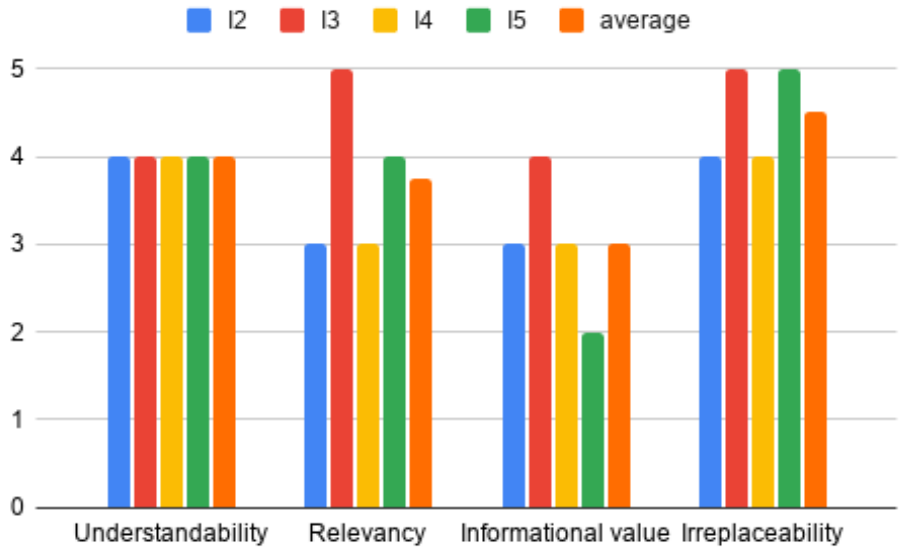

Figure 5.1: Evaluation results for prototype 1: Combining usage data with bug reports. Higher is better.

Understandability

Both of the prototypes were rated high in understandability, indicating that the information presented by the prototypes was easy to grasp. In particular, P2 achieved a perfect score from all of the interviewees. Error codes in P1 were a bit difficult, while the heatmap was seen as a very intuitive way to present usage data.

> *The visuality, as in the colors and the wholeness, it's like really easy to understand that there where the color is of lighter blue then it has been used very rarely and where it is red there has been used the most.* (I2)

Relevancy

There is some deviation in the relevancy of the prototypes. For P1, standard deviation is 0.83, and for P2, standard deviation is 0.5. The relatively big deviation of P1 is explained mainly by the fact that the interviewees are working with different tasks, where different kinds of information are relevant. Thus the error information conveyed by P1 is not of equal importance for everyone. On the other hand, all of the interviewees felt that knowing where the errors are occurring and how many users they are affecting is somewhat

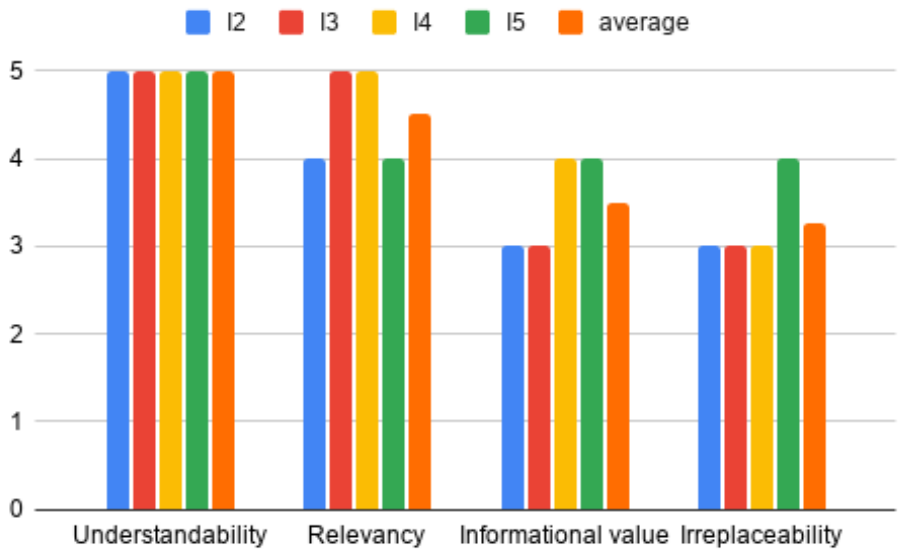

Figure 5.2: Evaluation results for prototype 2: Combining usage data with heatmaps. Higher is better.

important information regardless of their role. The consultant describes P1's relevancy from a non-programmer's perspective:

> *Because I am not a programmer I would probably say two because of the technical information like the error codes. But I'll increase the relevancy to three because I am interested in the end-user's viewpoint, so of course, it's important for me as well if the program is constantly crashing due to errors.* (I2)

As for P2, all of the developers had coincidentally worked on the same exact view that was selected for the prototype. This made them feel that it was interesting to finally see how a view that they have been part of developing is truly being used. The consultant also felt that the conveyed usage information was relevant because for their work it is important to know how the end-users are really making use of the software.

Informational value

Both P1 and P2 achieved similar average "information value" ratings of 3 and 3.5, with standard deviations of 0.63 and 0.5, respectively. For P1, the interviewees unanimously agreed that the informational value was only mediocre because the presented data is quite coarse-grained. Thus the interviewees were left longing for more fine-grained information

to answer more detailed questions such as *"What were the exact URLs?"*, and *"What groups of users have encountered these errors?"*

As for the informational value of P2, the interviewees were again almost unanimous. All of the developers explicitly mentioned that the information conveyed by the prototype is quite simple, as it only attaches actions to user-interface elements. Once again, this lack of more fine-grained information left the interviewees longing for more information:

> *From this I get the volume and the target, that is the data pair I can get from this. But I might have questions such as how many users, and what roles do the users have who have interacted with these user-interface elements?* (I5)

Irreplaceability

For the irreplaceability dimension, P1 obtained an average score of 4.5 with a standard deviation of 0.5. In other words, the interviewees found the information conveyed by the prototype to be very difficult or troublesome to obtain via other means. The interviewees explained that obtaining the data necessary for calculating the "Users affected" and "'Error rate" columns could be a tedious task:

> *Assuming that the error rate and users affected had to be calculated by hand, I would rate the irreplaceability as a five. Because if you have to dig for the data, you wouldn't be doing it.* (I5)

P2 scored an average of 3.25 on the irreplaceability dimension with a standard deviation of 0.43, indicating that the same information could be obtained via other means with some effort. The interviewees agreed it that would be possible to obtain the same information from the raw data. However, the developers who were aware of the technical implementation claimed that mapping the events to the interface elements would be a tedious task. Furthermore, all of the interviewees explicitly stated that the effectiveness and clarity of the heatmap presentation would be hard to achieve via other means. One developer also pointed out that the prototype could be replaced by conducting live user interviews:

> *This is a very efficient way to present the data. But I think that it might also be beneficial to conduct user interviews to see how the users really use the product. That way we could discuss with the users, understand what they are trying to do, and see where they get stuck. Besides, from this prototype, it is impossible to know how long it took for a user to find the correct button or input.* (I3)

Learned information and its practicality

The interviewees reported gaining some new insights from P1. Before seeing the prototype, the interviewees were already aware that there were some errors in the system. However, the developers reported that some of the error rates were surprisingly high and that the most error-prone URL was different than what they had anticipated. Furthermore, one developer mentioned that the categorization of error codes for each URL was helpful, as it allowed them to understand where the error might be originating from. Both the error rate and the ”users affected”-dimension were seen as interesting.

When asked how the newly learned information from P1 could be used in practice, two developers mentioned that it could be used by developers to self-direct their bug fixing efforts. Furthermore, the same two developers mentioned that such information could be used to justify taking a break from normal feature development only to fix bugs for a while. I4 also mentioned that the error rate could potentially be used as a goal to be worked towards:

> *It might be good to show this prototype to the development team during a weekly meeting and decide that for a week we should only focus on fixing errors stemming from a particular URL. We could set goals to define how low the error rate should be.* (I4)

For P2, the interviewees reported that they were able to grasp a good overview of how the presented part of the user interface is being used. Two of the interviewees reported that the high usage of the code filter was rather unexpected, raising further questions on why that is. P2 was also able to confirm one developer's assumptions on the low usage of some of the available features. Furthermore, one developer felt that seeing such a presentation was an astounding experience:

> *This concept is great, I have never seen this before. [...] It is really eye-opening to see this kind of visualization of a familiar program. [...] I would really like to see this method to be put to practice.* (I5)

All of the four interviewees identified that the aforementioned information learned from P2 could be primarily used to detect underused features. By detecting these underused features, discussion on the root cause of the issue could be started:

> *If we spot a feature that is experiencing low usage over the entire user population, then we could think why. Is it a useful feature, or is it difficult to grasp?* (I2)

5.1.4 Putting the prototypes to real use

The interviewees were unanimously in favor when asked if putting the prototypes to use would be a good idea. P2 especially was seen as particularly impactful by one of the developers:

> *I think that P2 could absolutely drive forward the transition towards a more fact-based decision-making process in the user-interface side of things.* (I5)

Implementation

When asked to consider how the prototypes could be merged into the development workflow, two of the developers immediately started imagining automatically updating dashboards. The rest of the interviewees went with a manual approach, where one of the developers would manually fetch the data and update the prototypes on-demand. The main driver behind the dashboard approach was that after setting it up, it would be always up-to-date and readily available:

> *If we, as a group, believe that the prototypes would be beneficial, then I think that automating it (the creation of prototypes) would be worth it. Because if someone had to construct them manually on-demand, then we would probably see the prototypes only a couple of times.* (I3)

No matter how the prototypes would be created, the consensus was that they should be taken a look at every 1–2 weeks. P1, portraying the error status of the system was deemed to be worth revisiting every week. P2, on the other hand, would be more suitable for ad-hoc use when a particular part of the user interface is the focus of the team's development efforts. Furthermore, the priority of this prototype-related work would depend on what had been decided as a team. A developer argued that the priority of the prototype related work would be higher than "regular development work":

> *If we have decided to implement such prototypes, then the priority of that work would be higher than regular development work. Because once the work on side of the prototypes is finished, we can then use them to better target our regular development efforts.* (I3)

I5 gives a similar statement:

> *If we really have the opportunity and a general consensus that we want to implement such an (automatic dashboard) system, then does it really make sense to*

> *keep doing "regular development work", or should we set up the system first? Then once the system would be ready to use we could look at the "regular work" from a fresh perspective with this new information and perhaps make new realizations and adjust our priorities. So in a way, this should affect the prioritization, and not just be a thing to be prioritized.* (I5)

Hindering and driving factors

Several driving and hindering factors became apparent during the interviews. The only hindering factor which appeared multiple times was higher priority tasks taking precedence. In fact, it was mentioned by each interviewee. Potential technical difficulties when opting for the dashboard approach were also apparent.

Softer issues were mentioned by two interviewees. Even if the whole team more or less agrees to try out the prototypes, everyone will still have their own opinions on the actual benefits brought forward by the prototypes. Furthermore, as time goes on and the initial boost of motivation fades, the utilization rate might slowly fade to inexistence.

The driving factors identified in the interviews were not uniform. One developer mentioned that looking at the prototypes should become a routine part of the group's activities. Another developer stressed the importance that everyone should take the prototypes seriously and that the team should have a common will to use them. Furthermore, another developer argued that in order to be successful, the prototypes would have to be easy to use and reliable:

> *The overall ease of use would help the prototype(s) succeed. It needs to be easy to use, automatically generated, and always up-to-date. In addition, it has to be reliable and robust so that when you need the prototype you can always get it.* (I5)

The final factor which might help with the success of such prototypes is the setting of goals and rewards. One interviewee pitched the idea of setting prototype-related goals, for example by defining that the error rate of the most error-prone URL should be less than 5% by next week. In addition, the developers could even be offered bonuses or other benefits to motivate them to work even harder towards the decided goal.

5.2 Validity

The author has an acknowledged personal bias towards thinking that software data utilization is in general something that has a lot of potential and should be utilized further where possible. This bias was fought against by having the interview questions checked by third parties and the questions were purposefully designed to include questions for both negative and positive aspects of the proposed methods/prototypes for putting software usage data to use.

The fact that the author had worked with some of the interview subjects in the past, may have had an impact on how the interviewees responded to the interview questions. For example, when the interviewees were asked if there was something "wrong/bad" in the prototype, the interviewees may have answered less harshly than if the prototype was made by a program or some other person than the interviewer. Thus the overall rating of the prototypes may be higher than it should be. This threat was fought against by informing the interviewees that the prototypes were created based on pre-existing literature. In the end, however, the results contained a lot of positives, but also a lot of negatives related to the prototypes constructed by the author. Thus it is reasonable to believe that the interviewees were quite open in expressing their honest thoughts and feelings towards the prototypes.

Furthermore, due to implementation, the quality of the usage data used for constructing the prototypes is not perfect. Consequently, the validity of the information conveyed by the prototypes is not perfect. Therefore some of the reportedly gained insights from the prototypes may be incorrect.

Finally, as the methods and prototypes were not actually put to practice, no conclusions on the actual feasibility and usefulness should be drawn.

6 Discussion

This chapter combines discussion and analysis of the results and provides some pointers for future work. We begin in Section 6.1 by analyzing our results by reflecting them on each other and existing literature. Next, in Section 6.2 we discuss our results in light of our three research questions. Finally, in Section 6.3 we give some pointers for future work in the domain of software usage data utilization.

6.1 Analysis

In the following subsections, we will be discussing the main findings of our study. Furthermore, we reflect our findings on related work.

6.1.1 Use cases of software usage data

By analyzing the relevant literature, we identified a wide range of use cases for utilizing software usage data. The most common use cases were related to better understanding the usage of the software and its users, but also in staying on top of the performance and quality of the software.

Furthermore, we noted that utilizing software usage data to help with software testing was a very uncommon use case in the literature. Whereas Buse and Zimmermann [18] found that both developers and managers are interested in information that could help in targeting testing efforts. Perhaps this discrepancy between the use case and the information needs of managers and developers is due to the lack of methods and tooling for employing software usage data in the context of software testing.

From the case study we found out that in practice, software usage data is used to fulfill a subset of the use cases identified in the literature. Both the developers and the managers had found software usage data to be beneficial in better understanding the usage of the system, further allowing them to locate issues and improvement needs. In other words, software usage data had acted as the first step towards enhancing the quality of the software. Some developers had also found software usage data to be useful in debugging hard to reproduce issues.

Furthermore, it was mentioned that software usage data can be used as a source of implicit user feedback [34]. When compared to traditional feedback collection methods, such as interviews or questionnaires, software usage data was seen as a non-biased and easily collectible alternative. While the traditional ways of feedback collection were seen as laborsome and easily biased. However, it was also observed that there are drawbacks to using software usage data as a source of feedback, as automatically collected data can be faulty, difficult to interpret, and lacking detail and context.

A multiple case study of three software development companies [15] made three observations that are relevant to our study 1) software usage data is rarely used for improving the current version of the product, 2) software usage data is mainly used for troubleshooting activities, and 3) software usage data does not often convey feature-level information.

Contrary to the first observation, in our case study, the usage data had been used explicitly to improve the current version of the product. This is, however, because there exists only one version of the product which is being continuously improved. Our case study results are in line with the second observation, in that software usage data is mainly used for troubleshooting (debugging) activities. Our case study's results differ on the third observation, as the usage data of our case study includes information on the usage of specific features. This difference can be explained by the types of software products being developed by the organizations, as Holmström et. al [15] studied embedded software, whereas we studied a web application.

6.1.2 Utilization methods of software usage data

In the semi-structured literature review, we identified a total of 9 methods for utilizing software usage data. Manual analysis with simple visualizations was clearly the most commonly utilized method for taking advantage of software usage data, while the rest of the methods appeared only a few times in the literature. This is likely because manually analyzing the data and creating simple visualizations is relatively easy, while for example, designing and implementing an ML/AI system for processing the data is more difficult and requires expertise.

From the case study, we found that the sole method used for utilizing the available software usage data was manual analysis. This is not surprising, as the overall utilization rate for the data was low. Thus, investing time and resources in utilizing more advanced methods for software usage data utilization could not be justified. Furthermore, we identified that

the usage of the data takes place at random intervals with no pre-defined procedure. Both the low utilization rate and the lack of a standardized procedure have been noted in previous work [15].

6.1.3 Practicality of the prototypes

In the case study, we evaluated two methods for utilizing software usage data 1) Combining usage data with bug reports, and 2) Visualizing software usage with heatmaps. Both of the methods were received with interest by both the managers and the developers. As previous work has identified [18], the developers and the managers were interested in different types of information, thus affecting the value of the methods on a personal level. However, even within the developers, there was significant variation on how relevant the information conveyed by the method was for the individual.

From the literature, we gathered that by combining usage data with bug reports the severity of the bugs could be better evaluated, thus helping in the prioritization process. In practice we found this claim to be true, as developers reported that such information could be used to justify focusing development efforts in fixing the most error-prone endpoints, and to help them self-direct their development efforts.

As for visualizing software usage with heatmaps, the literature indicated that such an approach could be especially valuable for new users to get familiar with the system. As no users of the system were interviewed, we can not accept nor reject this hypothesis. In practice, visualizing software usage with heatmaps was seen by the developers and the managers as an eye-opening and intuitive way to present an overview of the usage of a system. Furthermore, the presentation was found to be well suited for either confirming or falsifying assumptions on the usage of particular features and to detect underutilized features.

The proposed prototypes of the methods, however, were not perfect. Most importantly, the information presented by the prototypes was on a too high level and would have to be more fine-grained for the prototype to be actionable on its own. Besides, as previous work has identified, interpreting feature-level usage data can be difficult because there is no simple way to classify whether a feature is valuable or wasteful [15].

6.1.4 Success factors for software usage data utilization

From the case study, we found that methods for utilizing software usage data were seen as interesting, insightful, and valuable. However, despite the clear interest in the methods, there would still be some obstacles that need to be overcome for the methods to work efficiently and sustainably in a real-world setting.

For a software usage data utilization method to be successfully instilled into an organization or a development team, any work related to the method should be prioritized at least as high as regular development work. It might even be feasible to give the method-related work a higher priority than normal development work, as once the method is ready to be utilized, it can potentially help prioritize and focus normal development efforts.

The method would have to be accepted, taken seriously, and found useful by all the members of the development team. Furthermore, the technical implementation of the method would have to be feasible and robust. Finally, using the method would have to become a routine part of the team's regular workflow, for example by allocating five minutes for the method in every weekly meeting.

6.2 Research questions revisited

In this thesis, we set out to find answers to three research questions revolving around the idea of utilizing software usage data to help with software development. In the following, we will be revisiting each of the research questions one by one.

RQ1: How and for what purposes can software usage data be used for

In our first research question, we set out to understand how and for what purposes software usage data can be used for. We approached the question by conducting a semi-structured literature review and identified 9 concrete methods and 9 use cases for software usage data. We found that by a clear margin, the most common method for utilizing software usage data is via manual analysis and simple visualizations, likely due to the simplicity of the method. The rest of the identified methods were significantly more sophisticated and consequently appeared more rarely in the literature.

Furthermore, we found that software usage data can be used to both perform and help with a significant collection of software development-related activities. The focus of these

activities seems to lie in obtaining an enhanced understanding of both the usage of the system and its users. Furthermore, there appears to be a strong quality-related component, which aims to monitor and evaluate the state of the software, allowing for detecting and resolving issues in the system.

RQ2: How do developers and managers utilize software usage data in practice?

In our second research question, we sought to understand how software usage data is utilized in practice by interviewing the developers and a consultant of an agile software development team. In short, the usage data was utilized mainly for debugging and to obtain information on the usage of specific features. We found that the utilization of software usage data varied greatly within the development team. Some of the developers had found the data to be beneficial in their weekly work, especially when working on resolving hard to reproduce issues. On the other hand, some developers had not yet found any use for the data in their regular workflows. Furthermore, the usage data had also been of use for the developers while performing data-driven design to determine which features of a particular view should be removed and which should be kept, based on the usage of those features.

The consultant had found some use cases for the usage data as well. Unlike the developers, the consultant's interest lied in understanding the usage of the application on a high level. Furthermore, like the developers, the consultant was also interested in discovering underused features so that the root cause of the underuse could be found and the application be improved further.

RQ3: How do developers and managers experience methods for utilizing software usage data?

In our third and final research question, we aimed at understanding how developers and managers experience methods for utilizing software usage data. To this end, two prototypes of methods for utilizing software usage data were constructed based on existing literature and then evaluated by the developers and managers of a software development team.

The methods for utilizing software usage data were received with interest. Overall, both of the methods were rated highly by both the managers and the developers. Due to their simplicity, the methods were easy to understand but that came with the cost of decreased

informational value. In practice, the simplicity of the methods should likely be sacrificed to increase their informational value. Furthermore, the methods were deemed to be difficult to replace, as the information conveyed by the prototypes, especially if they were to be automated, as obtaining the same information manually was deemed to be too laborious. The relevancy of the methods was tied to each person's individual information needs, as identified in previous work [18].

Furthermore, we found that the overall attitude towards putting such software usage data utilization methods to use was positive. Both the developers and the managers felt that software usage data utilization methods should be employed to a greater extent. However, as discussed in Section 6.1.4 on page 43, putting the methods to real use would likely require overcoming some obstacles.

6.3 Future work

We were only able to experiment with and evaluate two of the nine identified methods for utilizing software usage data. Thus, as the groundwork of identifying the methods has been laid out, there lies an opportunity for experimenting and evaluating the rest of the methods in a similar manner.

Furthermore, it was unfortunately not possible to put the methods to real use and to evaluate their performance and feasibility in a real setting. It would be interesting to see how they truly perform, especially regarding extended usage ranging several months. What benefits could be achieved and what obstacles would have to be overcome?

It was noted that the process of generating heatmaps from feature-based usage data required a moderate amount of manual work. Thus to make the method (and other software usage data utilization methods) feasible, further research on the tooling for automating some or all of the heatmap generation-related tasks would be of value.

7 Conclusions

In this thesis, we set out to explore how software usage data can be used to assist developers and managers in software development-related activities. We approached the problem by combining the knowledge of existing literature with novel information from a case study. This chapter summarizes our research efforts.

We began by conducting a semi-structured literature review to identify methods and use cases for software usage data. We quickly learned that there exists a wide range of methods for putting software usage data to use, ranging from manual analysis to sophisticated AI/ML systems. We learned that via these methods, a collection of use cases focusing on gaining knowledge and resolving issues can be fulfilled.

Next, we conducted a case study where we sought to explore how a small development team utilizes software usage data in their work. We learned that even though usage data are being collected, it is not used very often. Furthermore, we found that debugging issues occurring in production environments is the dominant use case for software usage data.

Lastly, we sought to understand how do managers and developers experience methods for utilizing software usage data. To this end, we constructed and evaluated prototypes of two software usage data utilization methods with four members of an agile software development team. Both the managers and the developers were interested to see the prototypes in action with real usage data stemming from a familiar program. It became apparent that such methods produce information that would be difficult to obtain via other means. In other words, methods for utilizing software usage data can provide irreplaceable information that is relevant and useful for both managers and developers. Therefore, practitioners should consider introducing methods for utilizing software usage data in their development practices. Furthermore, we learned that presenting usage data in a coarse-grained and simple format can work well for igniting individuals' thought processes towards the root cause of the higher-level observation.

As a final remark, we noticed that even though methods for utilizing software usage data are received with interest, successfully putting them to use might be a different story as there are social, technological, and organizational factors that might hinder the success of such methods.

Bibliography

[1] N. Bunkley. "Joseph Juran, 103, Pioneer in Quality Control, Dies". In: *The New York Times. Business* (Mar. 3, 2008). ISSN: 0362-4331. URL: https://www.nytimes.com/2008/03/03/business/03juran.html (visited on 02/12/2021).

[2] T. Lehtonen, V. Eloranta, M. Leppänen, and E. Isohanni. "Visualizations as a Basis for Agile Software Process Improvement". In: *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*. 2013 20th Asia-Pacific Software Engineering Conference (APSEC). Vol. 1. Dec. 2013, pp. 495–502. DOI: 10.1109/APSEC.2013.71.

[3] A. Mattila, T. Lehtonen, H. Terho, T. Mikkonen, and K. Systä. "Mashing Up Software Issue Management, Development, and Usage Data". In: *2015 IEEE/ACM 2nd International Workshop on Rapid Continuous Software Engineering*. 2015 IEEE/ACM 2nd International Workshop on Rapid Continuous Software Engineering. May 2015, pp. 26–29. DOI: 10.1109/RCoSE.2015.12.

[4] "IEEE Standard Glossary of Software Engineering Terminology". In: *IEEE Std 610.12-1990* (Dec. 1990), pp. 1–84. DOI: 10.1109/IEEESTD.1990.101064.

[5] T. Ryan and S. Xenos. "Who Uses Facebook? An Investigation into the Relationship between the Big Five, Shyness, Narcissism, Loneliness, and Facebook Usage". In: *Computers in Human Behavior*. 2009 Fifth International Conference on Intelligent Computing 27.5 (Sept. 1, 2011), pp. 1658–1664. ISSN: 0747-5632. DOI: 10.1016/j.chb.2011.02.004. URL: http://www.sciencedirect.com/science/article/pii/S0747563211000379 (visited on 02/02/2021).

[6] V. Venkatesh and M. Morris. "Why Don't Men Ever Stop to Ask for Directions? Gender, Social Influence, and Their Role in Technology Acceptance and Usage Behavior". In: *MIS Quarterly: Management Information Systems* 24.1 (2000), pp. 115–136. DOI: 10.2307/3250981.

[7] *Slack's Outage on January 4th 2021*. Slack Engineering. Feb. 1, 2021. URL: https://slack.engineering/slacks-outage-on-january-4th-2021/ (visited on 02/01/2021).

[8] J. Johnson. “ROI, It’s Your Job!” In: *Published Keynote Third International Conference on Extreme Programming*. Vol. 4. 3. 2002, p. 48.

[9] M. Cohn. *Are 64% of Features Really Rarely or Never Used?* Mountain Goat Software. URL: https://www.mountaingoatsoftware.com/blog/are-64-of-features-really-rarely-or-never-used (visited on 01/22/2021).

[10] E. Juergens, M. Feilkas, M. Herrmannsdoerfer, F. Deissenboeck, R. Vaas, and K. Prommer. “Feature Profiling for Evolving Systems”. In: *2011 IEEE 19th International Conference on Program Comprehension*. 2011 IEEE 19th International Conference on Program Comprehension. June 2011, pp. 171–180. DOI: 10.1109/ICPC.2011.12.

[11] S. Eder, M. Junker, E. Jürgens, B. Hauptmann, R. Vaas, and K. Prommer. “How Much Does Unused Code Matter for Maintenance?” In: *2012 34th International Conference on Software Engineering (ICSE)*. 2012 34th International Conference on Software Engineering (ICSE). June 2012, pp. 1102–1111. DOI: 10.1109/ICSE.2012.6227109.

[12] B. W. Boehm and P. N. Papaccio. “Understanding and Controlling Software Costs”. In: *IEEE Transactions on Software Engineering* 14.10 (Oct. 1988), pp. 1462–1477. ISSN: 1939-3520. DOI: 10.1109/32.6191.

[13] J. Srivastava, R. Cooley, M. Deshpande, and P.-N. Tan. “Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data”. In: *ACM SIGKDD Explorations Newsletter* 1.2 (Jan. 1, 2000), pp. 12–23. ISSN: 1931-0145. DOI: 10.1145/846183.846188. URL: http://doi.org/10.1145/846183.846188 (visited on 01/27/2021).

[14] G. Zheng and S. Peltsverger. “Web Analytics Overview”. In: 2015. DOI: 10.4018/978-1-4666-5888-2.CH756.

[15] H. Holmström Olsson and J. Bosch. “Towards Data-Driven Product Development: A Multiple Case Study on Post-Deployment Data Usage in Software-Intensive Embedded Systems”. In: *Lean Enterprise Software and Systems*. Ed. by B. Fitzgerald, K. Conboy, K. Power, R. Valerdi, L. Morgan, and K.-J. Stol. Lecture Notes in Business Information Processing. Berlin, Heidelberg: Springer, 2013, pp. 152–164. ISBN: 978-3-642-44930-7. DOI: 10.1007/978-3-642-44930-7_10.

[16] T. H. Davenport, J. G. Harris, and R. Morison. *Analytics at Work: Smarter Decisions, Better Results*. Harvard Business Press, 2010. 231 pp. ISBN: 978-1-4221-7769-3. Google Books: 2otJuvfvflgC.

[17] T. Menzies and T. Zimmermann. "Software Analytics: So What?" In: *IEEE Software* 30.4 (July 2013), pp. 31–37. ISSN: 1937-4194. DOI: 10.1109/MS.2013.86.

[18] R. P. L. Buse and T. Zimmermann. "Information Needs for Software Development Analytics". In: *2012 34th International Conference on Software Engineering (ICSE)*. 2012 34th International Conference on Software Engineering (ICSE). June 2012, pp. 987–996. DOI: 10.1109/ICSE.2012.6227122.

[19] W. A. Association et al. "Web Analytics Definitions". In: *Digital Analytics Association* (2008).

[20] D. Zhang, S. Han, Y. Dang, J. Lou, H. Zhang, and T. Xie. "Software Analytics in Practice". In: *IEEE Software* 30.5 (Sept. 2013), pp. 30–37. ISSN: 1937-4194. DOI: 10.1109/MS.2013.94.

[21] F. Palomba, G. Bavota, M. D. Penta, R. Oliveto, D. Poshyvanyk, and A. D. Lucia. "Mining Version Histories for Detecting Code Smells". In: *IEEE Transactions on Software Engineering* 41.5 (May 2015), pp. 462–489. ISSN: 1939-3520. DOI: 10.1109/TSE.2014.2372760.

[22] D. Zhang, Y. Dang, J.-G. Lou, S. Han, H. Zhang, and T. Xie. "Software Analytics as a Learning Case in Practice: Approaches and Experiences". In: *Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering*. MALETS '11. New York, NY, USA: Association for Computing Machinery, Nov. 12, 2011, pp. 55–58. ISBN: 978-1-4503-1022-2. DOI: 10.1145/2070821.2070829. URL: http://doi.org/10.1145/2070821.2070829 (visited on 01/19/2021).

[23] R. Musson, J. Richards, D. Fisher, C. Bird, B. Bussone, and S. Ganguly. "Leveraging the Crowd: How 48,000 Users Helped Improve Lync Performance". In: *IEEE Software* 30.4 (July 2013), pp. 38–45. ISSN: 1937-4194. DOI: 10.1109/MS.2013.67.

[24] J. Bosch. "Building Products as Innovation Experiment Systems". In: *Software Business*. Ed. by M. A. Cusumano, B. Iyer, and N. Venkatraman. Lecture Notes in Business Information Processing. Berlin, Heidelberg: Springer, 2012, pp. 27–39. ISBN: 978-3-642-30746-1. DOI: 10.1007/978-3-642-30746-1_3.

[25] 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: *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*. 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications. Aug. 2014, pp. 9–16. DOI: 10.1109/SEAA.2014.75.

[26] J. Webster and R. T. Watson. "Analyzing the Past to Prepare for the Future: Writing a Literature Review". In: *MIS Quarterly* 26.2 (2002), pp. xiii–xxiii. ISSN: 0276-7783. JSTOR: 4132319.

[27] S. Martínez-Fernández, A. M. Vollmer, A. Jedlitschka, X. Franch, L. López, P. Ram, P. Rodríguez, S. Aaramaa, A. Bagnato, M. Choraś, and J. Partanen. "Continuously Assessing and Improving Software Quality With Software Analytics Tools: A Case Study". In: *IEEE Access* 7 (2019), pp. 68219–68239. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2917403.

[28] *Application Monitoring and Error Tracking Software*. Sentry. URL: https://sentry.io/welcome/ (visited on 04/14/2021).

[29] R. T. Fielding and J. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. RFC 7231. 2014. URL: https://www.rfc-editor.org/rfc/rfc7231.txt (visited on 05/20/2021).

[30] *Dynamic Heatmaps for the Web*. URL: https://www.patrick-wied.at/static/heatmapjs/ (visited on 04/14/2021).

[31] P. Runeson and M. Höst. "Guidelines for Conducting and Reporting Case Study Research in Software Engineering". In: *Empirical Software Engineering* (2008). DOI: 10.1007/s10664-008-9102-8.

[32] V. Braun and V. Clarke. "Using Thematic Analysis in Psychology". In: *Qualitative Research in Psychology* 3.2 (Jan. 1, 2006), pp. 77–101. ISSN: 1478-0887. DOI: 10.1191/1478088706qp063oa. URL: https://www.tandfonline.com/doi/abs/10.1191/1478088706qp063oa (visited on 04/06/2021).

[33] J. W. Creswell and D. L. Miller. "Determining Validity in Qualitative Inquiry". In: *Theory Into Practice* 39.3 (Aug. 1, 2000), pp. 124–130. ISSN: 0040-5841. DOI: 10.1207/s15430421tip3903_2. URL: https://doi.org/10.1207/s15430421tip3903_2 (visited on 04/06/2021).

[34] W. Maalej, H.-J. Happel, and A. Rashid. “When Users Become Collaborators: Towards Continuous and Context-Aware User Input”. In: *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*. OOPSLA ’09. New York, NY, USA: Association for Computing Machinery, Oct. 25, 2009, pp. 981–990. ISBN: 978-1-60558-768-4. DOI: 10.1145/1639950.1640068. URL: http://doi.org/10.1145/1639950.1640068 (visited on 06/05/2021).

Appendix A Sources of the semi-structured literature review

This reference listing contains the initial 99 sources used in the semi-structured literature review. Each reference ends in parentheses, containing the SID (source identifier) assigned for the source, or the text "Excluded" if the source was excluded.

B. Driscoll and Z. Zhao. "Automation of NERSC Application Usage Report". In: Proceedings of 2020 IEEE/ACM International Workshop on HPC User Support Tools, HUST 2020 and the Workshop on Programming and Performance Visualization Tools, ProTools 2020 - Held in Conjunction with SC 2020: The International Conference for High Performance Computing, Networking, Storage and Analysis. Institute of Electrical and Electronics Engineers Inc., 2020, pp. 10–18. ISBN: 978-0-7381-1070-7. DOI: 10.1109/HUSTProtools51951.2020.00009. *(SID: 1)*.

M. Bano, E. Groen, I. Hadar, and A. Mahmoud. "Welcome to the Fourth International Workshop on Crowd-Based Requirements Engineering (CrowdRE'20)". In: Proceedings - 4th International Workshop on Crowd-Based Requirements Engineering, CrowdRE 2020. Institute of Electrical and Electronics Engineers Inc., 2020, pp. VI–VII. ISBN: 978-1-72818-362-6. DOI: 10.1109/CrowdRE51214.2020.00005. *(SID: Excluded)*.

L. Andrade, P. Machado, and W. Andrade. "Can Operational Profile Coverage Explain Post-Release Bug Detection?" In: *Software Testing Verification and Reliability* 30.4-5 (2020). ISSN: 09600833. DOI: 10.1002/stvr.1735. *(SID: 2)*.

B. Sudan, S. Cansiz, E. Ogretici, and M. Aktas. "Prediction of Success and Complex Event Processing in E-Learning". In: 2nd International Conference on Electrical, Communication and Computer Engineering, ICECCE 2020. Institute of Electrical and Electronics Engineers Inc., 2020. ISBN: 978-1-72817-116-6. DOI: 10.1109/ICECCE49384.2020.9179281. *(SID: 3)*.

T. Zhang, B. Hartmann, M. Kim, and E. Glassman. "Enabling Data-Driven API Design with Community Usage Data: A Need-Finding Study". In: Conference on Human Factors in Computing Systems - Proceedings. Association for Computing Machinery, 2020. ISBN: 978-1-4503-6708-0. DOI: 10.1145/3313831.3376382. *(SID: 4)*.

W.-P. Brinkman, P. Gray, and K. Renaud. “Computer-Assisted Recording, Pre-Processing, and Analysis of User Interaction Data”. In: Proceedings of the 20th BCS HCI Group Conference: Engage, HCI 2006. British Computer Society HCI Group, 2020, pp. 273–275. *(SID: Excluded)*.

O. Dæhli, B. Kristoffersen, and T. Sandnes. “Lessons Learned from Developing and Evaluating an Educational Database Modeling Tool”. In: vol. 2020-October. Proceedings of the European Conference on E-Learning, ECEL. Academic Conferences and Publishing International Limited, 2020, pp. 129–138. ISBN: 978-1-912764-78-5. DOI: 10.34190/EEL.20.055. *(SID: 5)*.

A. Murad, N. Hyde, S. Chang, R. Lederman, R. Bosua, M. Pirotta, R. Audehm, C. Yates, A. Briggs, A. Gorelik, C. Chiang, and J. Wark. “Quantifying Use of a Health Virtual Community of Practice for General Practitioners’ Continuing Professional Development: A Novel Methodology and Pilot Evaluation”. In: *Journal of Medical Internet Research* 21.11 (2019). ISSN: 14388871. DOI: 10.2196/14545. *(SID: 6)*.

J. Johanssen, A. Kleebaum, B. Bruegge, and B. Paech. “How Do Practitioners Capture and Utilize User Feedback during Continuous Software Engineering?” In: vol. 2019-September. Proceedings of the IEEE International Conference on Requirements Engineering. IEEE Computer Society, 2019, pp. 153–164. ISBN: 978-1-72813-912-8. DOI: 10.1109/RE.2019.00026. *(SID: 7)*.

M. Stade, S. Scherr, P. Mennig, F. Elberzhager, and N. Seyff. “Don’t Worry, Be Happy - Exploring Users’ Emotions during App Usage for Requirements Engineering”. In: vol. 2019-September. Proceedings of the IEEE International Conference on Requirements Engineering. IEEE Computer Society, 2019, pp. 375–380. ISBN: 978-1-72813-912-8. DOI: 10.1109/RE.2019.00048. *(SID: 8)*.

A. Rhoads and W. Hill. “Development of a Tool for Estimating the Life Cycle Climate Performance of MAC Systems”. In: *SAE Technical Papers* 2019-April (April 2019). ISSN: 01487191. DOI: 10.4271/2019-01-0611. *(SID: Excluded)*.

C. Sahin, L. Pollock, and J. Clause. “Supporting Software Evolution through Feedback on Executing/Skipping Energy Tests for Proposed Source Code Changes”. In: *Journal of Software: Evolution and Process* 31.4 (2019). ISSN: 20477481. DOI: 10.1002/smr.2155. *(SID: Excluded)*.

M. Barta, S. Schimanski, J. Buchhorn, and A. Nawrot. "Development of a Web Based Framework to Objectively Compare and Evaluate Software Solutions". In: *Advances in Intelligent Systems and Computing* 787 (2019), pp. 3–11. ISSN: 21945357. DOI: 10.1007/978-3-319-94229-2_1. *(SID: 9).*

M. Cosar. "Carbon Footprint in Data Center: A Case Study". In: *Fresenius Environmental Bulletin* 28.2 (2019), pp. 600–607. ISSN: 10184619. *(SID: Excluded).*

A. Kleyner and D. Elmore. "Weibull Analysis and Zero-Time Failures. What Are Your Data Analysis Options?" In: vol. 2019-January. Proceedings - Annual Reliability and Maintainability Symposium. Institute of Electrical and Electronics Engineers Inc., 2019. ISBN: 978-1-5386-6554-1. DOI: 10.1109/RAMS.2019.8769298. *(SID: Excluded).*

S. Kosić. "The efficacy analysis of the university of rijeka library's book aquisition [Analiza uspješnosti kupnje knjiga za fond sveučilišne knjižnice rijeka]". In: *Vjesnik Bibliotekara Hrvatske* 62.2 (2019), pp. 131–148. ISSN: 05071925. DOI: 10.30754/vbh.62.2.771. *(SID: Excluded).*

M. Oriol, M. Stade, F. Fotrousi, S. Nadal, J. Varga, N. Seyff, A. Abello, X. Franch, J. Marco, and O. Schmidt. "FAME: Supporting Continuous Requirements Elicitation by Combining User Feedback and Monitoring". In: Proceedings - 2018 IEEE 26th International Requirements Engineering Conference, RE 2018. Institute of Electrical and Electronics Engineers Inc., 2018, pp. 217–227. ISBN: 978-1-5386-7418-5. DOI: 10.1109/RE.2018.00030. *(SID: 10).*

M. Arthur. "Managing a Comprehensive Cost-per-Use Project in a Large Academic Library". In: *Serials Review* 44.4 (2018), pp. 299–306. ISSN: 00987913. DOI: 10.1080/00987913.2018.1558936. *(SID: Excluded).*

J. Fernandes, I. Brunton, G. Strudwick, S. Banik, and J. Strauss. "Physician Experience with Speech Recognition Software in Psychiatry: Usage and Perspective". In: *BMC Research Notes* 11.1 (2018). ISSN: 17560500. DOI: 10.1186/s13104-018-3790-y. *(SID: 11).*

L. Poppe, C. Van Der Mispel, G. Crombez, I. De Bourdeaudhuij, H. Schroé, and M. Verloigne. "How Users Experience and Use an eHealth Intervention Based on Self-Regulation: Mixed-Methods Study". In: *Journal of Medical Internet Research* 20.10 (2018). ISSN: 14388871. DOI: 10.2196/10412. *(SID: Excluded).*

R. Braga, P. Neto, R. Rabêlo, J. Santiago, and M. Souza. "A Machine Learning Approach to Generate Test Oracles". In: ACM International Conference Proceeding Series. Association for Computing Machinery, 2018, pp. 142–151. ISBN: 978-1-4503-6503-1. DOI: 10.1145/3266237.3266273. *(SID: 12)*.

G. Orsini, D. Bade, and W. Lamersdorf. "CloudAware: Empowering Context-Aware Self-Adaptation for Mobile Applications". In: *Transactions on Emerging Telecommunications Technologies* 29.4 (2018). ISSN: 21615748. DOI: 10.1002/ett.3210. *(SID: Excluded)*.

J. Johanssen, A. Kleebaum, B. Bruegge, and B. Paech. "Feature Crumbs: Adapting Usage Monitoring to Continuous Software Engineering". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 11271 LNCS (2018), pp. 263–271. ISSN: 03029743. DOI: 10.1007/978-3-030-03673-7_19. *(SID: 13)*.

S. Yaman, F. Fagerholm, M. Munezero, H. Maenpaa, and T. Mannisto. "Notifying and Involving Users in Experimentation: Ethical Perceptions of Software Practitioners". In: vol. 2017-November. International Symposium on Empirical Software Engineering and Measurement. IEEE Computer Society, 2017, pp. 199–204. ISBN: 978-1-5090-4039-1. DOI: 10.1109/ESEM.2017.31. *(SID: 14)*.

Z. Coker, K. Damevski, C. Le Goues, N. Kraft, D. Shepherd, and L. Pollock. "Behavior Metrics for Prioritizing Investigations of Exceptions". In: Proceedings - 2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017. Institute of Electrical and Electronics Engineers Inc., 2017, pp. 554–563. ISBN: 978-1-5386-0992-7. DOI: 10.1109/ICSME.2017.62. *(SID: 15)*.

"Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering and Education Track, ICSE-SEET 2017". In: Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering and Education Track, ICSE-SEET 2017. Institute of Electrical and Electronics Engineers Inc., 2017. ISBN: 978-1-5386-2671-9. *(SID: Excluded)*.

M. Jamieson, B. O'Neill, B. Cullen, M. Lennon, S. Brewster, and J. Evans. "ForgetMeNot: Active Reminder Entry Support for Adults with Acquired Brain Injury". In: vol. 2017-May. Conference on Human Factors in Computing Systems - Proceedings. Association for Computing Machinery, 2017, pp. 6012–6023. ISBN: 978-1-4503-4655-9. DOI: 10.1145/3025453.3025888. *(SID: 16)*.

M. Ahmed, A. Mohamed, R. Homod, H. Shareef, and K. Khalid. "Awareness on Energy Management in Residential Buildings: A Case Study in Kajang and Putrajaya". In: *Journal of Engineering Science and Technology* 12.5 (2017), pp. 1280–1294. ISSN: 18234690. *(SID: Excluded).*

A. Streicher, W. Roller, and C. Biegemeier. "Application of Adaptive Game-Based Learning in Image Interpretation". In: Proceedings of the 11th European Conference on Games Based Learning, ECGBL 2017. Academic Conferences and Publishing International Limited, 2017, pp. 975–978. ISBN: 978-1-911218-56-2. *(SID: 17).*

J. Piacenza, S. Mayoral, B. Albarhami, and S. Lin. "Understanding the Importance of Post Occupancy Usage Trends during Concept-Stage Sustainable Building Design". In: vol. 4. Proceedings of the ASME Design Engineering Technical Conference. American Society of Mechanical Engineers (ASME), 2017. ISBN: 978-0-7918-5816-5. DOI: 10.1115/DETC2017-67461. *(SID: Excluded).*

J. Fletcher and W. Malalasekera. "Development of a User-Friendly, Low-Cost Home Energy Monitoring and Recording System". In: *Energy* 111 (2016), pp. 32–46. ISSN: 03605442. DOI: 10.1016/j.energy.2016.05.027. *(SID: Excluded).*

A. Stocker and J. Müller. "Exploring Use and Benefit of Corporate Social Software: Measuring Success in the Siemens Case References+". In: *Journal of Systems and Information Technology* 18.3 (2016), pp. 277–296. ISSN: 13287265. DOI: 10.1108/JSIT-03-2016-0021. *(SID: 18).*

M. Javad Nasiri, A. Chirani, M. Amin, R. Halabian, and A. Imani Fooladi. "Isoniazid-Resistant Tuberculosis in Iran: A Systematic Review". In: *Tuberculosis* 98 (2016), pp. 104–109. ISSN: 14729792. DOI: 10.1016/j.tube.2016.03.007. *(SID: Excluded).*

J. Garcia and A. Paiva. "Maintaining Requirements Using Web Usage Data". In: vol. 100. Procedia Computer Science. Elsevier B.V., 2016, pp. 626–633. DOI: 10.1016/j.procs.2016.09.204. *(SID: 19).*

"Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE". In: vol. 2016-January. Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE. Knowledge Systems Institute Graduate School, 2016. ISBN: 1-891706-39-X 978-1-891706-39-4. *(SID: Excluded).*

C. Macpherson. "How M&V Makes a Difference - A Real World Case Study". In: vol. 1. 39th World Energy Engineering Conference, WEEC 2016. AEE Energy Books, 2016, pp. 442–448. ISBN: 978-1-5108-3076-9. *(SID: Excluded).*

J. Garcia and A. Paiva. "An Automated Approach for Requirements Specification Maintenance". In: *Advances in Intelligent Systems and Computing* 444 (2016), pp. 827–833. ISSN: 21945357. DOI: 10.1007/978-3-319-31232-3_78. *(SID: Excluded).*

W. Maalej, M. Nayebi, T. Johann, and G. Ruhe. "Toward Data-Driven Requirements Engineering". In: *IEEE Software* 33.1 (2016), pp. 48–54. ISSN: 07407459. DOI: 10.1109/MS.2015.153. *(SID: 20).*

J. Garcia and A. Paiva. "REQAnalytics: A Recommender System for Requirements Maintenance". In: *International Journal of Software Engineering and its Applications* 10.1 (2016), pp. 129–140. ISSN: 17389984. DOI: 10.14257/ijseia.2016.10.1.13. *(SID: Excluded).*

"1st International Conference on Smart Trends in Information Technology and Computer Communications, SmartCom 2016". In: *Communications in Computer and Information Science* 628 CCIS (2016), pp. 1–918. ISSN: 18650929. *(SID: Excluded).*

F. Doyle, M.-J. Duarte, and J. Cosgrove. "Design of an Embedded Sensor Network for Application in Energy Monitoring of Commercial and Industrial Facilities". In: vol. 83. Energy Procedia. Elsevier Ltd, 2015, pp. 504–514. DOI: 10.1016/j.egypro.2015.12.170. *(*SID: Excluded).

D. Shepherd, K. Damevski, and L. Pollock. "How and When to Transfer Software Engineering Research via Extensions". In: vol. 2. Proceedings - International Conference on Software Engineering. IEEE Computer Society, 2015, pp. 239–240. ISBN: 978-1-4799-1934-5. DOI: 10.1109/ICSE.2015.152. *(SID: Excluded).*

"Proceedings - 2nd International Workshop on Rapid Continuous Software Engineering, RCoSE 2015". In: Proceedings - 2nd International Workshop on Rapid Continuous Software Engineering, RCoSE 2015. Institute of Electrical and Electronics Engineers Inc., 2015. ISBN: 978-1-4799-1934-5. *(SID: Excluded).*

R. Geng and J. Tian. "Improving Web Navigation Usability by Comparing Actual and Anticipated Usage". In: *IEEE Transactions on Human-Machine Systems* 45.1 (2015), pp. 84–94. ISSN: 21682291. DOI: 10.1109/THMS.2014.2363125. *(SID: 21).*

S. Sivagnanam, A. Majumdar, K. Yoshimoto, V. Astakhov, A. Bandrowski, M. Martone, and N. Carnevale. "Early Experiences in Developing and Managing the Neuroscience Gateway". In: *Concurrency Computation* 27.2 (2015), pp. 473–488. ISSN: 15320626. DOI: 10.1002/cpe.3283. *(SID: 22).*

R. MacIntyre, J. Alcock, P. Needham, and J. Lambert. "Measuring the Usage of Repositories via a National Standards-Based Aggregation Service: IRUS-UK". In: New Avenues for Electronic Publishing in the Age of Infinite Collections and Citizen Science: Scale, Openness and Trust - Proceedings of the 19th International Conference on Electronic Publishing, Elpub 2015. IOS Press BV, 2015, pp. 83–92. ISBN: 978-1-61499-561-6. DOI: 10.3233/978-1-61499-562-3-83. *(SID: Excluded).*

L. Hokkanen and K. Väänänen-Vainio-Mattila. "UX Work in Startups: Current Practices and Future Needs". In: *Lecture Notes in Business Information Processing* 212 (2015), pp. 81–92. ISSN: 18651348. DOI: 10.1007/978-3-319-18612-2_7. *(SID: 23).*

R. Poston and A. Calvert. "Vision 2020: The Future of Software Quality Management and Impacts on Global User Acceptance". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9191 (2015), pp. 748–760. ISSN: 03029743. DOI: 10.1007/978-3-319-20895-4_70. *(SID: 24).*

J. Pepper, A. Zhang, R. Li, and X. Wang. "Usage Results of a Mobile App for Managing Urinary Incontinence". In: *Journal of Urology* 193.4 (2015), pp. 1292–1297. ISSN: 00225347. DOI: 10.1016/j.juro.2014.10.009. *(SID: 25).*

C. Piovesan. "A Real Time Chemical Monitoring Platform for Inventory and Usage". In: vol. 2015-January. Proceedings - SPE Annual Technical Conference and Exhibition. Society of Petroleum Engineers (SPE), 2015, pp. 804–810. ISBN: 978-1-5108-1322-9. DOI: 10.2118/174780-ms. *(SID: Excluded).*

V. Kiberu, J. Matovu, F. Makumbi, C. Kyozira, E. Mukooyo, and R. Wanyenze. "Strengthening District-Based Health Reporting through the District Health Management Information Software System: The Ugandan Experience". In: *BMC Medical Informatics and Decision Making* 14.1 (2014). ISSN: 14726947. DOI: 10.1186/1472-6947-14-40. *(SID: 26).*

T. Evans, W. Barth, J. Browne, R. Deleon, T. Furlani, S. Gallo, M. Jones, and A. Patra. "Comprehensive Resource Use Monitoring for HPC Systems with TACC Stats". In: Proceedings of HUST 2014: 1st International Workshop on HPC User Support Tools - Held in Conjunction with SC 2014: The International Conference for High Performance Computing, Networking, Storage and Analysis. Institute of Electrical and Electronics Engineers Inc., 2014, pp. 13–21. ISBN: 978-1-4799-7023-0. DOI: 10.1109/HUST.2014.7. *(SID: Excluded).*

H. Liu, Y. Liu, G. Xue, and Y. Gao. "Case Study on Software Refactoring Tactics". In: *IET Software* 8.1 (2014), pp. 1–11. ISSN: 17518806. DOI: 10.1049/iet-sen.2012.0121. *(SID: Excluded)*.

W. Li, J. Matejka, T. Grossman, and G. Fitzmaurice. "Deploying Communitycommands: A Software Command Recommender System Case Study". In: vol. 4. Proceedings of the National Conference on Artificial Intelligence. AI Access Foundation, 2014, pp. 2922–2929. ISBN: 978-1-57735-680-6. *(SID: 27)*.

W. Snipes, A. Nair, and E. Murphy-Hill. "Experiences Gamifying Developer Adoption of Practices and Tools". In: 36th International Conference on Software Engineering, ICSE Companion 2014 - Proceedings. Association for Computing Machinery, 2014, pp. 105–114. ISBN: 978-1-4503-2768-8. DOI: 10.1145/2591062.2591171. *(SID: 28)*.

K. Dullemond, B. Van Gameren, M.-A. Storey, and A. Van Deursen. "Fixing the 'out of Sight out of Mind' Problem: One Year of Mood-Based Microblogging in a Distributed Software Team". In: IEEE International Working Conference on Mining Software Repositories. 2013, pp. 267–276. ISBN: 978-1-4673-2936-1. DOI: 10.1109/MSR.2013.6624038. *(SID: 29)*.

T. Furlani, B. Schneider, M. Jones, J. Towns, D. Hart, S. Gallo, R. Deleon, C.-D. Lu, A. Ghadersohi, R. Gentner, A. Patra, G. Laszewski, F. Wang, J. Palmer, and N. Simakov. "Using XDMoD to Facilitate XSEDE Operations, Planning and Analysis". In: ACM International Conference Proceeding Series. 2013. ISBN: 978-1-4503-2170-9. DOI: 10.1145/2484762.2484763. *(SID: Excluded)*.

S. Khatoon, A. Mahmood, G. Li, and J. Xu. "A Novel Integrated Framework to Increase Software Quality by Mining Source Code". In: *Journal of Software Engineering* 7.3 (2013), pp. 86–105. ISSN: 18194311. DOI: 10.3923/jse.2013.86.105. *(SID: Excluded)*.

J. Matejka, T. Grossman, and G. Fitzmaurice. "Patina: Dynamic Heatmaps for Visualizing Application Usage". In: Conference on Human Factors in Computing Systems - Proceedings. 2013, pp. 3227–3236. ISBN: 978-1-4503-1899-0. DOI: 10.1145/2470654.2466442. *(SID: 30)*.

J.-Y. Oh and A. Uggirala. "Data-Driven Design Process in Adoption of Marking Menus for Large Scale Software". In: vol. 2013-April. Conference on Human Factors in Computing Systems - Proceedings. Association for Computing Machinery, 2013, pp. 2327–2330. ISBN: 978-1-4503-1899-0. DOI: 10.1145/2468356.2468757. *(SID: 31)*.

A. Fry. “A Hybrid Model for Managing Standard Usage Data: Principles for e-Resource Statistics Workflows”. In: *Serials Review* 39.1 (2013), pp. 21–28. ISSN: 00987913. DOI: 10.1016/j.serrev.2012.12.001. *(SID: Excluded)*.

H. Olsson and J. Bosch. “Post-Deployment Data Collection in Software-Intensive Embedded Products”. In: *Lecture Notes in Business Information Processing* 150 (2013), pp. 79–89. ISSN: 18651348. DOI: 10.1007/978-3-642-39336-5_9. *(SID: 32)*.

D. Haun, A. Foley, and P. Jarreau. “Development and Feasibility of an Electronic White Blood Cell Identification Trainer.” In: *Clinical laboratory science : journal of the American Society for Medical Technology* 26.1 (2013), pp. 23–29. ISSN: 0894959X. DOI: 10.29074/ascls.26.1.23. *(SID: 33)*.

“2012 9th IEEE Working Conference on Mining Software Repositories, MSR 2012 - Proceedings”. In: IEEE International Working Conference on Mining Software Repositories. 2012. ISBN: 978-1-4673-1761-0. *(SID: Excluded)*.

“PLATEAU’11 - Proceedings of the 3rd ACM SIGPLAN Workshop on Evaluation and Usability of Programming Languages and Tools”. In: PLATEAU’11 - Proceedings of the 3rd ACM SIGPLAN Workshop on Evaluation and Usability of Programming Languages and Tools. 2011. ISBN: 978-1-4503-1024-6. *(SID: Excluded)*.

T. Salfischberger, I. Van De Weerd, and S. Brinkkemper. “The Functional Architecture Framework for Organizing High Volume Requirements Management”. In: 2011 5th International Workshop on Software Product Management, IWSPM 2011 - Part of the 19th IEEE International Requirements Engineering Conference. 2011, pp. 17–25. ISBN: 978-1-4577-1147-3. DOI: 10.1109/IWSPM.2011.6046208. *(SID: Excluded)*.

J. Du, Y. Yang, Z. Lin, Q. Wang, M. Li, and F. Yuan. “A Case Study on Usage of a Software Process Management Tool in China”. In: Proceedings - Asia-Pacific Software Engineering Conference, APSEC. 2010, pp. 443–452. ISBN: 978-0-7695-4266-9. DOI: 10.1109/APSEC.2010.57. *(SID: 34)*.

P. Lew, L. Olsina, and L. Zhang. “Integrating Quality, Quality in Use, Actual Usability and User Experience”. In: 2010 6th Central and Eastern European Software Engineering Conference, CEE-SECR 2010. 2010, pp. 117–123. ISBN: 978-1-4577-0606-6. DOI: 10.1109/CEE-SECR.2010.5783161. *(SID: 35)*.

M. Funk, P. Hoyer, and S. Link. “Model-Driven Instrumentation of Graphical User Interfaces”. In: Proceedings of the 2nd International Conferences on Advances in Computer-

Human Interactions, ACHI 2009. 2009, pp. 19–25. ISBN: 978-0-7695-3529-6. DOI: 10.1109/ACHI.2009.16. *(SID: 36)*.

S. Hickey, C. Fitzpatrick, M. O'connell, and M. Johnson. "Use Phase Signals to Promote Lifetime Extension for Windows PCs". In: *Environmental Science and Technology* 43.7 (2009), pp. 2544–2549. ISSN: 0013936X. DOI: 10.1021/es8020638. *(SID: Excluded)*.

S. Bateman, C. Gutwin, N. Osgood, and G. McCalla. "Interactive Usability Instrumentation". In: EICS'09 - Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems. Association for Computing Machinery, 2009, pp. 45–54. ISBN: 978-1-60558-600-7. DOI: 10.1145/1570433.1570443. *(SID: 37)*.

Q. Zhang and R. Segall. "Web Mining: A Survey of Current Research, Techniques, and Software". In: *International Journal of Information Technology and Decision Making* 7.4 (2008), pp. 683–720. ISSN: 02196220. DOI: 10.1142/S0219622008003150. *(SID: Excluded)*.

M. Jones. "Polymorphism and Page Tables: Systems Programming from a Functional Programmer's Perspective". In: Proceedings of the ACM SIGPLAN International Conference on Functional Programming, ICFP. 2008, p. 265. ISBN: 978-1-59593-919-7. DOI: 10.1145/1411204.1411207. *(SID: Excluded)*.

R. García and C. Kloos. "Web Usage Mining in a Blended Learning Context: A Case Study". In: Proceedings - The 8th IEEE International Conference on Advanced Learning Technologies, ICALT 2008. 2008, pp. 982–984. ISBN: 978-0-7695-3167-0. DOI: 10.1109/ICALT.2008.229. *(SID: 38)*.

O. McGrath. "Seeking Activity: On the Trail of Users in Open and Community Source Frameworks". In: Proceedings ACM SIGUCCS User Services Conference. 2007, pp. 234–239. ISBN: 978-1-59593-634-9. DOI: 10.1145/1294046.1294103. *(SID: 39)*.

J. Miller, B. Friedman, G. Jancke, and B. Gill. "Value Tensions in Design: The Value Sensitive Design, Development, and Appropriation of a Corporation's Groupware System". In: GROUP'07 - Proceedings of the 2007 International ACM Conference on Supporting Group Work. 2007, pp. 281–290. ISBN: 978-1-59593-845-9. DOI: 10.1145/1316624.1316668. *(SID: 40)*.

K. Atalağ, S. Bilgen, G. Gür, and S. Boyacioğlu. "Evaluation of the Turkish Translation of the Minimal Standard Terminology for Digestive Endoscopy by Development of an Endoscopic Information System". In: *Turkish Journal of Gastroenterology* 18.3 (2007), pp. 157–164. ISSN: 13004948. *(SID: 41)*.

A. Birukou, E. Blanzieri, V. D'Andrea, P. Giorgini, and N. Kokash. "Improving Web Service Discovery with Usage Data". In: *IEEE Software* 24.6 (2007), pp. 47–54. ISSN: 07407459. DOI: 10.1109/MS.2007.169. *(SID: 42)*.

K.-T. Lam and D. Chan. "Building an Institutional Repository: Sharing Experiences at the HKUST Library". In: *OCLC Systems and Services* 23.3 (2007), pp. 310–323. ISSN: 1065075X. DOI: 10.1108/10650750710776440. *(SID: 43)*.

R. Girardi and L. Balby Marinho. "A Domain Model of Web Recommender Systems Based on Usage Mining and Collaborative Filtering". In: *Requirements Engineering* 12.1 (2007), pp. 23–40. ISSN: 09473602. DOI: 10.1007/s00766-006-0038-5. *(SID: Excluded)*.

F. Zhang, B. Yang, W. Song, and N. Li. "Intelligent Decision Support System Based on Data Mining: Foreign Trading Case Study". In: 2007 IEEE International Conference on Control and Automation, ICCA. Institute of Electrical and Electronics Engineers Inc., 2007, pp. 1487–1491. ISBN: 1-4244-0818-0 978-1-4244-0818-4. DOI: 10.1109/ICCA.2007.4376609. *(SID: 44)*.

"7th IFIP International Conference on E-Business, e-Services, and e-Society, I3E 2007". In: *IFIP Advances in Information and Communication Technology* 252 VOLUME 2 (2007). ISSN: 18684238. *(SID: Excluded)*.

A. Lindoso and R. Girardi. "The SRAMO Technique for Analysis and Reuse of Requirements in Multi-Agent Application Engineering". In: WER 2006 - 9th Workshop on Requirements Engineering. 2006, pp. 41–50. *(SID: Excluded)*.

L. Sokvitne. "Redesigning the Opac: Moving Outside the Ilms". In: *Australian Academic and Research Libraries* 37.4 (2006), pp. 246–259. ISSN: 00048623. DOI: 10.1080/00048623.2006.10755344. *(SID: Excluded)*.

R. Girardi and A. Lindoso. "An Ontology-Driven Technique for the Architectural and Detailed Design of Multi-Agent Frameworks". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 3529 LNAI (2006), pp. 124–139. ISSN: 03029743. DOI: 10.1007/11916291_9. *(SID: Excluded)*.

J.-P. Norguet, E. Zimányi, and R. Steinberger. "Semantic Analysis of Web Site Audience". In: vol. 1. Proceedings of the ACM Symposium on Applied Computing. Association for Computing Machinery, 2006, pp. 525–529. ISBN: 1-59593-108-2 978-1-59593-108-5. DOI: 10.1145/1141277.1141401. *(SID: Excluded)*.

R. Girardi and A. Lindoso. "DDEMAS: A Domain Design Technique for Multi-Agent Domain Engineering". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 3770 LNCS (2005), pp. 141–150. ISSN: 03029743. DOI: 10.1007/11568346_16. *(SID: Excluded).*

J. Thevenet. "AMR System Helps Santa Fe Conserve Water". In: *Water and Wastewater International* 20.6 (2005), p. 23. ISSN: 08915385. *(SID: Excluded).*

Y. Zhang, H. Zhu, and S. Greenwood. "Website Complexity Metrics for Measuring Navigability". In: Proceedings - Fourth International Conference on Quality Software, QSIC 2004. 2004, pp. 172–179. ISBN: 0-7695-2207-6. *(SID: Excluded).*

J. Gómez, G. Kassem, C. Rautenstrauch, and M. Melato. "Analysis of User's Behaviour in Very Large Business Application Systems with Methods of the Web Usage Mining - A Case Study on SAP® R/3®". In: vol. 2663. Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science). Springer Verlag, 2003, pp. 329–338. ISBN: 3-540-40124-5 978-3-540-40124-7. DOI: 10.1007/3-540-44831-4_34. *(SID: 45).*

M. Prieto and M. Sicilia. "Designing Adaptive Mobile Applications: Abstract Components and Composite Behaviors". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 2795 (2003), pp. 378–383. ISSN: 03029743. DOI: 10.1007/978-3-540-45233-1_32. *(SID: 46).*

M. Hawley, P. O'Neill, L. Webb, and C. Roast. "A Provision Framework and Data Logging Tool to Aid the Prescription of Electronic Assistive Technology". In: *Technology and Disability* 14.2 (2002), pp. 43–52. ISSN: 10554181. DOI: 10.3233/tad-2002-14201. *(SID: 47).*

M. Christel. "Experiences with an Interactive Video Code Inspection Laboratory". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 640 LNCS (1992), pp. 395–411. ISSN: 03029743. DOI: 10.1007/3-540-55963-9_65. *(SID: 48).*

G. Boyd, A. Keller, and R. Kenner. "Remedial and Second Language English Teaching Using Computer Assisted Learning". In: *Computers and Education* 6.1 (1982), pp. 105–112. ISSN: 03601315. DOI: 10.1016/0360-1315(82)90019-7. *(SID: 49).*

W. Jessop, J. Kane, S. Roy, and J. Scanlon. "ATLAS-an Automated Software Testing System". In: Proceedings - International Conference on Software Engineering. IEEE Computer Society, 1976, pp. 629–635. *(SID: Excluded).*