COMBINING USER FEEDBACK AND MONITORING DATA TO SUPPORT EVIDENCE-BASED SOFTWARE EVOLUTION

Farnaz Fotrousi

Blekinge Institute of Technology
Doctoral Dissertation Series No. 2020:04

Department of Software Engineering

Combining User Feedback and Monitoring Data to Support Evidence-Based Software Evolution

Farnaz Fotrousi

Blekinge Institute of Technology Doctoral Dissertation Series
No 2020:04

Combining User Feedback and Monitoring Data to Support Evidence-Based Software Evolution

Farnaz Fotrousi

Doctoral Dissertation in
Software Engineering

Department of Software Engineering
Blekinge Institute of Technology
SWEDEN

2020 Farnaz Fotrousi
Department of Software Engineering
Publisher: Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden
Printed by Exakta Group, Sweden, 2020
ISBN: 978-91-7295-402-1
ISSN: 1653-2090
urn:nbn:se:bth-19397

To my parents, forever in my heart!

Abstract

Context: Companies continuously explore their software systems to acquire evidence for software evolution, such as bugs in the system and new functional or quality requirements. So far, managers have made decisions about software evolution based on evidence gathered from interpreting user feedback and monitoring data collected separately from software in use. These evidence-collection processes are usually unmethodical, lack a systematic guide, and have practical issues. This lack of a systematic approach leaves unexploited opportunities for detecting evidence for system evolution.

Objective: The main research objective is to improve evidence collection from software in use and guide software practitioners in decision-making about system evolution. Understanding useful approaches to collect user feedback and monitoring data, two important sources of evidence, and combining them are key objectives as well.

Method: We proposed a method for gathering evidence from software in use (GESU) using design-science research. We designed the method over three iterations and validated it in the European case studies FI-Start, Supersede, and Wise-IoT. To acquire knowledge for the design, we conducted further research using surveys and systematic mapping methods.

Results: The results show that GESU is not only successful in industrial environments but also yields new evidence for software evolution by bringing user feedback and monitoring data together. This combination helps software practitioners improve their understanding of end-user needs and system drawbacks, ultimately supporting continuous requirements elicitation and product evolution. GESU suggests monitoring a software system based on its goals to filter relevant data (i.e., goal-driven monitoring) and gathering user feedback when the system requests feedback about the software in use (i.e., system-triggered user feedback). The system identifies interesting situations of system use and issues automated requests for user feedback to interpret the evidence from user perspectives. We justified using goal-driven monitoring and system-triggered user feedback with complementary findings of the thesis. That showed the goals and characteristics of software systems constrain monitoring data. We thus narrowed the monitoring and observational focus on data aligned with goals instead of a massive amount of potentially useless data. Finally, we found that requesting feedback from users with a simple feedback form is a useful approach for motivating users to provide feedback.

Conclusion: Combining user feedback and monitoring data is helpful to acquire insights into the success of a software system and guide decision-making regarding its evolution. This work can be extended in the future by implementing an adaptive system for gathering evidence from combined user feedback and monitoring data.

Acknowledgments

I would like to express my sincere appreciation to my supervisors; Prof. Dr Samuel A. Fricker, Prof. Dr Markus Fiedler, and Prof. Dr Jürgen Börstler for their continuous, invaluable, and helpful support and guidance in my research. Despite their busy schedules, they were always available to share their deep knowledge and provide me with their insightful feedback on my study.

Thanks to my colleagues in DIPT and DIKO departments at BTH, and FHNW university for the enjoyable and educating conversations we have had. Especially, I would like to thank Deepika Badampudi for her continuous support and friendship. Also, Muhammad Usman and Melanie Stade for their supportive discussions and their kindness.

I also use this opportunity to express my gratitude to partners involved in the FI-STAR, Supersede, and Wise-IoT projects collaborated kindly and provided me with the opportunity of running these studies.

My deep appreciation goes to my family for being so encouraging and supportive. Especially, I am so grateful to my spouse, Shahryar, for his great emotional and technical support along the way. He was truly my invaluable consultant, who continuously shared with me his knowledge and insight. Arvid, my son, is responsible for some of the sweetest moments in my life in the past two and a half years, helping me to overcome the common frustrations along with my researches. Thank you, my son!

At last, special gratitude to my parents, for the life-long devotion, love, and support they bestowed on me. They will always have a share in the bests that I will achieve through my life.

Preface

Papers included in this thesis: The compilation thesis includes the following seven papers:

Chapter 2. **F. Fotrousi**, M. Stade, N. Seyff, S. Fricker, M. Fiedler (2020). “How do Users Characterise Feedback Features of an Embedded Feedback Channel?” – Submitted to a Journal.

Chapter 3. **F. Fotrousi**, S. Fricker, M. Fiedler (2018). “The Effect of Requests for User Feedback on Quality of Experience”, Software Quality Journal, 26(2), 385-415. DOI: 10.1007/s11219-017-9373-7.

Chapter 4. **F. Fotrousi**, S. Fricker, M. Fiedler (2014). “KPIs in Software Ecosystem: A Systematic Mapping Study”, 5th International Conference on the Software Business (ICSOB), Paphos, Cyprus: Springer, pp 194-211. DOI: 10.1007/978-3-319-08738-2.

Chapter 5. **F. Fotrousi**, S. Fricker (2016). “Software Analytics for Planning Product Evolution”, 7th International Conference of Software Business (ICSOB), Ljubljana, Slovenia: Springer, pp. 16-31. DOI: 10.1007/978-3-319-40515-5_2.

Chapter 6. **F. Fotrousi**, S. Fricker, M. Fiedler (2014). “Quality Requirements Elicitation based on Inquiry of Quality-Impact Relationships”, 22nd International Conference on Requirements Engineering (RE), Karlskrona, Sweden: IEEE, pp: 303-312. DOI: 10.1109/RE.2014.6912272.

Chapter 7. M. Oriol, M. Stade, **F. Fotrousi**, S. Nadal, J. Varga, N. Seyff, A. Abello, X. Franch, J. Marco, O. Schmidt (2018). “FAME: Supporting Continuous Requirements Elicitation by Combining User Feedback and Monitoring”, 26th International Conference on Requirements Engineering (RE), Banff, Canada: IEEE. pp: 217-227. DOI: 10.1109/RE.2018.00030.

Chapter 8. **F. Fotrousi**, S. Fricker, M. Fiedler, D. Wüest (2020). “A Method for Gathering Evidence from Software in Use to Support Software Evolution”- Submitted to a Journal.

Contribution Statement:

Farnaz Fotrousi was the main driver of the studies in Chapters 2, 3, 4, 5, 6, and 8 in designing, executing and reporting the studies. In the study presented in chapter 7, Farnaz contributed by designing a feedback gathering system and lead a team of students to implement the design. She also proposed a technical solution for a combination of user feedback and monitoring data. Farnaz Fotrousi contributed in writing particularly for related works and the description of FAME Framework in the user feedback parts. She reviewed and commented on the final draft.

Other contributions related to this thesis:

1. Contribution to deliverables of FI-STAR European project:
   D6.2: Common test platform
   D6.4: Validated services at experimentation sites

2. Contribution to deliverables of SUPERSEDE European project:
   D1.2: Direct multi-modal feedback gathering techniques, V1
   D1.4: Comprehensive monitoring techniques, v1

Related papers not included in this thesis:

1. **F. Fotrousi**, S. Fricker (2016). "QoE probe: A requirement-monitoring tool", REFSQ Workshops, co-located with the 22nd International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ), Gothenburg, Sweden: CEUR-WS.

2**. F. Fotrousi**, N. Seyff, J. Börstler (2017). "Ethical Considerations in Research on User Feedback", 25th International Requirements Engineering Conference Workshops (REW), Lisbon, Portugal: IEEE, pp. 194-198.

3. D. Wüest, **F. Fotrousi**, S. A. Fricker (2019). "Combining Monitoring and Autonomous Feedback Requests to Elicit Actionable Knowledge of System Use", 25nd International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ), Essen, Germany: Springer: pp. 209-225.

4. M. Stade, **F. Fotrousi**, N. Seyff, and O. Albrecht (2017). "Feedback Gathering from an Industrial Point of View", 25th International Conference on Requirements Engineering (RE), Lisbon, Portugal IEEE, pp. 71-79.

5. S. Fricker, K. Schneider, **F. Fotrousi**, C. Thuemmler (2016). "Workshop Videos for Requirements Communication", Requirements Engineering Journal, 21(4), pp. 521-552.

6. M. Stade, M. Oriol, O. Cabrera, **F. Fotrousi**, R. Schaniel, N. Seyff, O. Schmidt (2017). "Providing A User Forum Is Not Enough: first

experiences of a software company with CrowdRE", 25th International Requirements Engineering Conference Workshops (REW), Lisbon, Portugal: IEEE, pp. 164-169.

7. N. Seyff, M. Stade, **F. Fotrousi**, M. Glinz, E. Guzman, M. Kolpondinos-Huber, R. Schaniel (2017). "End-user Driven Feedback Prioritization", REFSQ Workshops, co-located with the 22nd International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ), Essen, Germany: CEUR-WS, pp. 1-7.

8. **F. Fotrousi**, K. Izadyan, and S. A. Fricker (2013). "Analytics for Product Planning: In-depth Interview Study with SaaS Product Managers." Sixth International Conference on Cloud Computing (CLOUD), Santa Clara Marriott, CA, USA: IEEE, pp. 871-879.

9. **F. Fotrousi** (2016). "Quality-Impact Assessment of Software Systems". In Ph.D. Symposium of 24th conference on Requirements Engineering Conference (RE), Beijing, China: IEEE, pp. 427-431.

10. S. Fricker, **F. Fotrousi**, M. Fiedler, P. Cousin (2013). "Quality of Experience Assessment based on Analytics", 2nd European Teletraffic Seminar (ETS), Karlskrona, Sweden.

11. J. Molleri, I. Nurdiani, **F. Fotrousi**, K. Petersen (2019). "Experiences on studying Attention through EEG in the Context of Review Tasks", Evaluation and Assessment in Software Engineering Conference (EASE), Copenhagen, Denmark: ACM. pp. 313-318.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Part 1

Kappa

Chapter 1 : Overview

1. Introduction

Software companies' products evolve by timely functionality changes, environmental adaptations, and performance and maintenance improvement (Rajlich and Bennett 2000; Taentzer et al. 2019). Software evolution brings products closer to customers' desires and needs, addresses competition pressure, and generates value for software companies (Lehman and Ramil 2003; Thew and Sutcliffe 2018). Continuously observing software in use and collecting user opinions allows software practitioners, such as developers, testers, requirements engineers, and product managers, to collect evidence about unresolved issues and unsatisfied user needs. They may decide to add new features or even remove existing ones when the risks of keeping them are higher than their generated values (Fabijan et al. 2016). Such evolution decisions consider constraints such as market saturation and political and legal concerns (Godfrey and German 2008).

Evidence is a body of facts and information that is interpreted to derive knowledge (Kitchenham et al. 2015). Evidence can reveal clues and guide software practitioners to seek hidden clues. Practitioners interpret evidence and merge it with their observations and experience to make evidence-based decisions about software evolution (Devanbu et al. 2016). Evidence is collected and organised hierarchically. The evidence for claims such as "the user did not like feature x" and "feature x has a response time of 10 seconds" can aggregate and support the decision to "improve feature x". For simplicity, this thesis refers to evidence for software evolution simply as *evidence*. *Evidence* can be gathered from several sources, including consultation with stakeholders, close observation of software and its environment, market research, and findings from the literature.

The two forms of gathering evidence – consulting stakeholders and closely observing software in use – are the foci of the thesis. Many techniques exist for consulting stakeholders, including workshops (Phaal et al. 2007), focus groups (Krueger 2009), and surveys (Fowler 2009). This thesis studies user feedback mainly via feedback forms embedded in software, which shortens the time between the feedback and the actual user experience and facilitates gathering immediate user perceptions.

Gathering evidence from close observation of a software system in use is usually performed using monitoring tools that constantly record logs. Monitoring systems allows software practitioners to continuously determine the degree to which products are successful to meet requirements and generate value for their company (Carreño and Winbladh 2013). Several monitoring tools and frameworks exist (Vierhauser et al. 2016) that make reports, including monitoring data in the form of measurements carried out over periods of time, which is sometimes referred to as *analytic. Analytics* is: the quantitative *measurement* of an entity relevant to a software product (Davenport and Harris 2007) that provides insight and actionable information (Zhang et al. 2011). This thesis mainly uses the term *monitoring data* but sometimes the term *analytics* to focus on the actionable characteristics of information.

"There is a lesson here. Extracting value from information is not primarily a matter of how much data you have or what technologies you use to analyze it, though these can help. Instead, it's how aggressively you exploit these resources and how much you use them to create new and better approaches to doing business."
(Davenport and Harris 2007)

So far, approaches to gathering evidence from user feedback and monitoring data have been unmethodical, have lacked systematic guidance, and have some practical issues. Stade et al. (2017) found that software practitioners were aware of the importance of user feedback and provided feedback channels but did not fully exploit the potential of user feedback for development and evolution. They added that practitioners were not always satisfied with the quality and quantity of the feedback received; speculatively, perhaps their feedback-gathering approach did not motivate users to give feedback or most feedback did not address particular issues under investigation. Most practitioners collect user feedback traditionally (i.e., passively). For example, they do not usually solicit feedback from users about new features and only hope that users react and give feedback when problems emerge. Software practitioners also do not gather user feedback systematically or educate their users to provide helpful feedback (Pagano and Brügge 2013). These reasons motivate further studies to improve gathering evidence from user feedback.

System monitoring also usually generates massive amounts of data that are not necessarily useful. Software practitioners often do not know how to find evidence in such data, perhaps needing a sophisticated data-mining approach to extract it. This motivates further studies to improve gathering evidence from monitoring systems by focusing, for example, on the process of gathering data instead of analysing them.

Lacking a systematic approach to improving evidence-gathering from systems in use disables software practitioners to rely on evidence from user feedback and monitoring data. Instead, they must rely on their own observations and experience to make decisions regarding software system evolution. Therefore, practitioners cannot assure that user demands are satisfied. User dissatisfaction increases user churn, meaning that users discontinue using the software system, consequently endangering the software's sustainability.

This PhD thesis aims to improve gathering evidence from software in use. The thesis is organised in eight chapters. We use design science to develop and evaluate method GESU (Gathering Evidence from Software in Use). GESU focusses on the process and activities of collecting user feedback, monitoring data, and combining both sources. The method was designed in three iterations and validated in case studies from the European projects FI-Start, Supersede and Wise-IoT, which are presented in Chapters 6, 7, and 8, respectively. To understand how to design GESU, we investigated how user feedback and monitoring data should be gathered using embedded tools in a software system.

Chapter 2 starts with reviewing embedded feedback channels, in a software system and their supported feedback features that allows giving feedback in different formats such as *natural language text* or *spoken language.* A feedback channel is referred as feedback tool or alternatively feedback form in this thesis. Then, Chapter 2 investigates the perceived characteristics of feedback features, such as ease of use and the capability for explaining a complicated situation, and also studies whether the characteristics have an impact on the use of the feedback channel. End users have different needs when providing feedback (Almaliki et al. 2014; Maalej et al. 2009). Some prefer to send feedback in the form of text or star ratings, whereas others prefer recording their screens and audio as feedback. Some users initiate feedback communication by pushing a feedback button (Morales-Ramirez et al. 2015), and others provide feedback when requested (Dennis et al. 2008). The former approach is called *push* or *user-triggered* and the latter *pull* or *system-triggered*. In Chapter 3, we study the second. We model a user feedback request and investigate whether and under which conditions the request for user feedback would disturb users.

Chapter 4 provides an overview of the literature to understand the main objectives of software managers in monitoring systems and the analytics they use to manage software ecosystems. Key performance indicators (KPIs) are those among the many important analytics, which are easily measurable and defined based on managers' objectives. Chapter 5 focusses on product planning and studying the analytics that managers find useful, including what factors influence the choice of analytics.

In summary, the thesis contributes to knowledge on improving gathering user feedback and monitoring data and combining them to support the evidence-based evolution of software systems.

The remainder of this chapter is structured as follows. Section 2 introduces the necessary background and related works. Sections 3 and 4 present this work's research questions and methods. Section 5 provides an overview of other chapters and summarises the results and main findings while answering the research questions. In section

6, we synthesise the findings by discussing this work's contributions, roadmap, and future work. Section 7 concludes the chapter.

2. Background and Related Work

This section explores the literature to provide context for the thesis's contributions.

2.1. Evidence-Based Software Evolution

Software evolution is a response to requests for new features, the existence of new platforms, and the desire to improve software quality and functionality while preventing issues such as market saturation, political and legal concerns, and software complexity (Godfrey and German 2008). Lehman and his colleagues have explored the software-evolution field, and formed a set of observations called the *laws of evolution* (Lehman 1980; Lehman and Ramil 2003). The laws are determined for software systems embedded in the real world, are produced by software teams for its users. Examples of the laws, indicating that systems become progressively more complex and less satisfying to users over time, reflect the need for continuous evolution, where feedback, either from humans or systems, is a driver (Lehman 1996). Additionally, Lehman et al. (1997) recommended exploiting observation of metrics and establishing baselines of key measures over time.

Several studies (Madhavji et al. 2006; Pagano and Brügge 2013) have reinforced that data collected from observing systems and feedback, referred to here as *evidence*, connote the idea of evolution decisions within the system environment. Bringing evidence to decisions about software evolution is defined as evidence-based software evolution, borrowed from evidence-based software engineering (Kitchenham et al. 2004).

Kitchenham et al. (2004) brought the concept of evidence-based decision-making from medicine and adapted it to software engineering to support decision-makers in both science and engineering fields. The way Kitchenham and her colleagues defined *evidence-based* focused on research evidence collected via primary studies through experiments and case studies as well as secondary studies through systematic literature reviews. The reviews aggregate primary studies and report objective summaries of evidence within the studies (Kitchenham et al. 2015). The research evidence fits software evolution for new technology and trend practices (Dyba et al. 2005). However, for feature- and quality-related evolution, such as requests for new features and bug removal, the evidence from real use observation of a specific software system should replace such research evidence.

Recent years have seen the practice of evidence-based software evolution emerge, such as the lean start-up approach (Blank 2013) and DevOps (i.e., software development and technology operations (Sjøberg et al. 2003). These practices are common in a build-monitor-learn loop. In essence, in an iterative approach, a software system is built, evidence from testing and monitoring the system as well as user feedback about the system are gathered, analysed, and interpreted, and changes are applied in a new loop. Such a loop allows shortening product development cycles and releasing them faster with more reliability. A systematic approach to gathering user feedback, monitoring data, and combining user feedback and monitoring data is still missing, however.

2.2. Gathering Evidence from User Feedback

User feedback communicates information about users' interests, needs, and how they are satisfied with a system (Knauss et al. 2009). User feedback can be provided either explicitly by users or implicitly by monitoring various user activities such as browsing, reading, and bookmarking (Lee and Brusilovsky 2009). The system users who provide feedback or are observed implicitly can be not only end-users but also developers, testers, or any other companies' stakeholders.

Several feedback tools and approaches have been designed to allow communicating user feedback. Feedback tools are either standalone or are embedded into systems (Fotrousi and Fricker 2016; Seyff et al. 2014). The feedback tools trigger feedback forms either by user request, such as pressing a feedback button, or by system request, such as an automatic pop-up window. The terms push and user-triggered refer to the former feedback approach and pull and system-triggered to the latter (Maalej et al. 2009). Such feedback forms enable users to communicate bug reports, feature requests, and praise (Maalej and Nabil 2015).

Feedback may be collected as a simple or a combination of free text, selected categories, ratings, and/or screenshots with annotations (Elling et al. 2012; Morales-Ramirez et al. 2015). In telecommunication domain *Rating* is a simple and common approach to measure *quality of experience (QoE), defined as "degree of delight or annoyance"* for evaluating services (Le Callet et al. 2012) with the scale called Mean-Opinion-Score (MOS)(ITU-T 2003).

Regardless of the design of feedback forms, several studies have described the challenges of analysing and interpreting user feedback, especially when contextual information is missing (Gottesdiener 2002).

2.3. Gathering Evidence from Monitoring Data

Monitoring a system use allows engineers to determine whether and to what degree the implemented system meets the requirements of its users during runtime (Carreño and Winbladh 2013). The insertion of code or sensors into a running system allows developers to continuously check the system's health, observe users, record their activities, and study the system's behaviour (Wellsandt et al. 2014). Observing the system and its quality at runtime, such as system performance, and its availability allows engineers evaluating system health and improving quality of services (QoS) (Wang et al. 2010). Observing user activities, such as sequences of feature usage, duration, and other contexts, enables requirements engineers to understand user needs better (Maalej et al. 2016). Such monitoring enables engineers to detect requirements violations, such as system failures, and react quickly to evolve the system (Leucker and Schallhart 2009).

The terms *monitoring data* and *analytics* interchangeably refer to sources of information or evidence that guide managers in their decisions. It is known as the *data-centric* style of decision-making (Buse and Zimmermann 2010) that includes measurements to generate data and transform them into indicators for decision support. In other words, analytics is the use of statistics from measurement characteristics (Davenport and Harris 2007) to obtain insight and actionable information (Zhang et al. 2011) and make data-driven decisions (Buse and Zimmermann 2010; Buse and Zimmermann 2012).

Several approaches have been studied to monitor systems or their requirements at runtime (Rabiser et al. 2017; Vierhauser et al. 2016). System use are monitored continuously or event-based (e.g., a user action like playing a song) and logs are recorded. (Fotrousi and Fricker 2016; Inzinger et al. 2014; Oriol et al. 2018; van Hoorn et al. 2009), Matomo (www.matomo.org), and Google Analytics (accounts.google.com), Mixpanel (www.mixpanel.com) are examples of such monitoring tools. There are other studies in which the system use is monitored based on requirements or a goal model (Goldsby et al. 2008; Qian et al. 2018; Wang et al. 2009). The benefit of the second approach is that data gathering is more focused, and a lighter analysis is introduced to find evidence for changes when compared to the first approach.

2.4. Combining Evidence from User Feedback and Monitoring Data

A number of researchers have proposed using both user feedback and monitoring data. For instance, (Dzvonyar et al. 2016) combined feedback data with monitoring data from the same end users who provided the feedback (e.g., log data). However, using this approach, the authors could only capture the data of end users who provided feedback, not data from other end users (e.g., to identify how many end users experienced an issue reported in feedback) or other types of monitoring data (e.g., quality of service [QoS]). In contrast, another approach (Dąbrowski et al. 2017) used monitoring data from all end users and applied process-mining techniques to observe their behaviour and elicit new requirements. The authors suggested that such information could be combined with feedback to refine the requirements and help improve the requirements-prioritisation process. However, they did not explore this research direction in-depth, leaving most to future work. MyExperience (Froehlich et al. 2007) is another solution that combines monitoring data and user feedback, and it is used to support studies on human behaviour or health (e.g., monitoring health-related metrics through sensors and asking end users how they feel). However, to the best of our knowledge, no generic solution has advanced from the conceptual stage to a technically implemented framework that comprehensively combines feedback gathering and monitoring to support continuous software system evolution.

3. Research Objectives and Questions

The study aims to improve GESU by combining monitoring data (i.e., system analytics) and user feedback. To achieve this goal, the thesis has the following objectives:

- OBJ1: Understanding approaches to gathering user feedback from software in use to support evidence-based software evolution
    - OBJ1.1: Understanding the characteristics of feedback features in an embedded feedback channel
    - OBJ1.2: Understanding how the characteristics of feedback features affect the use of its feedback channel
    - OBJ1.3: Understanding the strengths and limitations of various feedback features
    - OBJ1.4: Understanding the effect of requests for user feedback on experience of feedback senders
- OBJ2: Understanding approaches to monitoring the use of software systems to support evidence-based software evolution
    - OBJ2.1: Understanding the software system analytics that product managers use to manage software systems
    - OBJ2.2: Understanding how system analytics are used to plan product evolution
- OBJ3: Designing an effective method that combines user feedback and monitoring data from a software system in use to support evidence-based software evolution
- OBJ4: Validating the proposed method in real-world practice
    - OBJ4.1: Identifying how the proposed method is applicable in real-world practice
    - OBJ4.2 Specifying whether the method can be useful for software engineers in gathering and sharing knowledge for evolving software systems
    - OBJ4.3 Identifying how the method can be used to explain knowledge creation

Table 1-1. Research questions

Research Questions	Obj.	Chapters							Contrib.
		2	3	4	5	6	7	8	
RQ1. How can we gather user feedback of software systems in use to support evidence-based evolution?	OBJ1	*	*						C3, C4, C5, C6
RQ1.1. How do feedback senders characterise various feedback features in an embedded feedback channel?	OBJ 1.1, OBJ 1.3	*							C3
RQ1.2. What is the relationship between the characteristics of a feedback feature and the use of the feedback channel?	OBJ 1.2	*							C3
RQ1.3. Does a request for user feedback affect the perceived quality of software?	OBJ 1.4		*						C4, C5
RQ2. How can we gather monitoring data from software in use to support evidence-based evolution?	OBJ 2			*	*				C7, C8, C6
RQ2.1. What monitoring data do companies collect from software in use?	OBJ 2.1			*					C7
RQ2.2. How are monitoring data used to plan product evolution?	OBJ 2.2				*				C8
RQ3. How can we effectively gather evidence from software in use to support software engineers in their evolution decisions?	OBJ 3, OBJ 4					*	*	*	C1, C6, C2
RQ3.1. How can combining user feedback and monitoring data from software in use be applicable in real cases?	OBJ 4.1					*	*	*	C1, C2
RQ3.2. To what extent is the proposed method useful for software practitioners in their decision-making?	OBJ 4.1, OBJ 4.2					*	*	*	C1, C2
RQ 3.3. How well can the proposed method explain knowledge creation?	OBJ 4.3							*	C1

*: Objectives **: Contributions. ***: The labels of the research questions (RQ) and objectives (OBJ) are independent from the labels used in the studied papers, where each paper follows its own numbering schema.

Table 1-1 lists the study's research questions mapped to the corresponding objectives and the chapters that answer them. We also map each research question and its objective(s) to the corresponding contribution(s) of this thesis, which will be discussed later in Section 6.1.

4. Research Approach

This thesis follows a design-science approach (Hevner et al. 2004) with the primary goal of designing a method for GESU. Figure 1-1 presents the research framework with references to this paper's chapters. We iterated the GESU design and investigated the method's performance in context (Wieringa 2014). We received requirements from the product teams in each case study (the box labelled *Environment* in Figure 1-1) and applied concepts and technical solutions from the acquired knowledge base (the box labelled *Knowledge base* in Figure 1-1). We summarise and elaborate on the requirements in Section 4.1 and then explain cases in Chapters 6–8. We built the knowledge base with the studies presented in Chapters 2–5. We explain knowledge-creation theory as part of the knowledge base in Chapter 8. We present the GESU evaluation in Chapters 6–8. The first iteration relies on descriptive evaluation based on informed arguments and scenarios. For the second and third iterations, we use observational evaluation using case studies (Hevner et al. 2004). At a later stage, we use the same method to revisit the knowledge base in Chapter 8.

We explain our design-science approach in the next section (Section 4.1). To evaluate the proposed method during the design process, we use case studies (Section 4.2). To build the knowledge base of the design, we use systematic mapping (Section 4.3) and survey (Section 4.4) research methods.

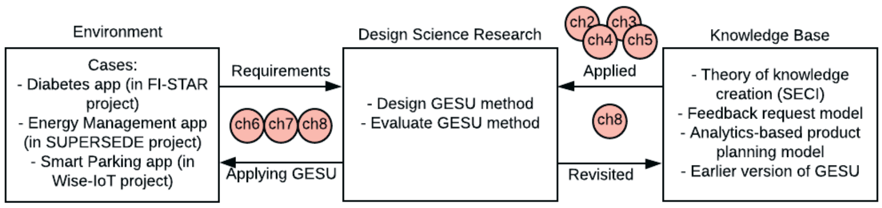

Environment
Cases:
- Diabetes app (in FI-STAR project)
- Energy Management app (in SUPERSEDE project)
- Smart Parking app (in Wise-IoT project)
Requirements
ch6
ch7
ch8
Applying GESU
Design Science Research
- Design GESU method
- Evaluate GESU method
ch2
ch3
ch4
ch5
Applied
ch8
Revisited
Knowledge Base
- Theory of knowledge creation (SECI)
- Feedback request model
- Analytics-based product planning model
- Earlier version of GESU

Figure 1-1. Research framework in the thesis*

*: red circles refer to chapters where the parts are discussed

4.1. Design Science

The core of this thesis is the use of design-science research to design and evaluate GESU over three iterations. In each iteration, we investigated problems from previous studies and designed the GESU method using our knowledge base. We then applied the method in a use case and conducted an empirical evaluation.

In the first iteration, there was a gap in the literature on how to gather and combine knowledge from running software to determine appropriate levels of good-enough system quality that on one side satisfy users and on the other side utilizes resources efficiently. Meeting the right level of quality was the requirement of the Diabetes case owner to balance benefits and cost. The right level of quality can guide the evolving quality requirements.

In the second iteration, we were missing a generic framework for gathering user feedback and system monitoring to address the requirements of the Energy efficiency management app for the configurable and continuous gathering of user feedback and monitoring data. We designed and implemented the framework, and particularly used in the Energy app to support them for continuous requirements elicitation by combining user feedback and usage monitoring.

In the third iteration, the case owner required to evaluate and improve the user adherence to the given recommendation for route and parking spots, in order to provide an insight on what and why should be changed. The solution needs to focus on monitoring the goal of users' adherence, instead of collecting massive data that are not potentially usefulness, and also engage users to share their perspective. None of the previous studies had proposed technical and theoretical solutions. We designed the solution following the theoretical framework for creating and sharing knowledge while we combined the findings from another chapter of the thesis (Chapters 2-5). The evaluation phase involved both evaluation of SECI and applicability and usefulness of the method for the case.

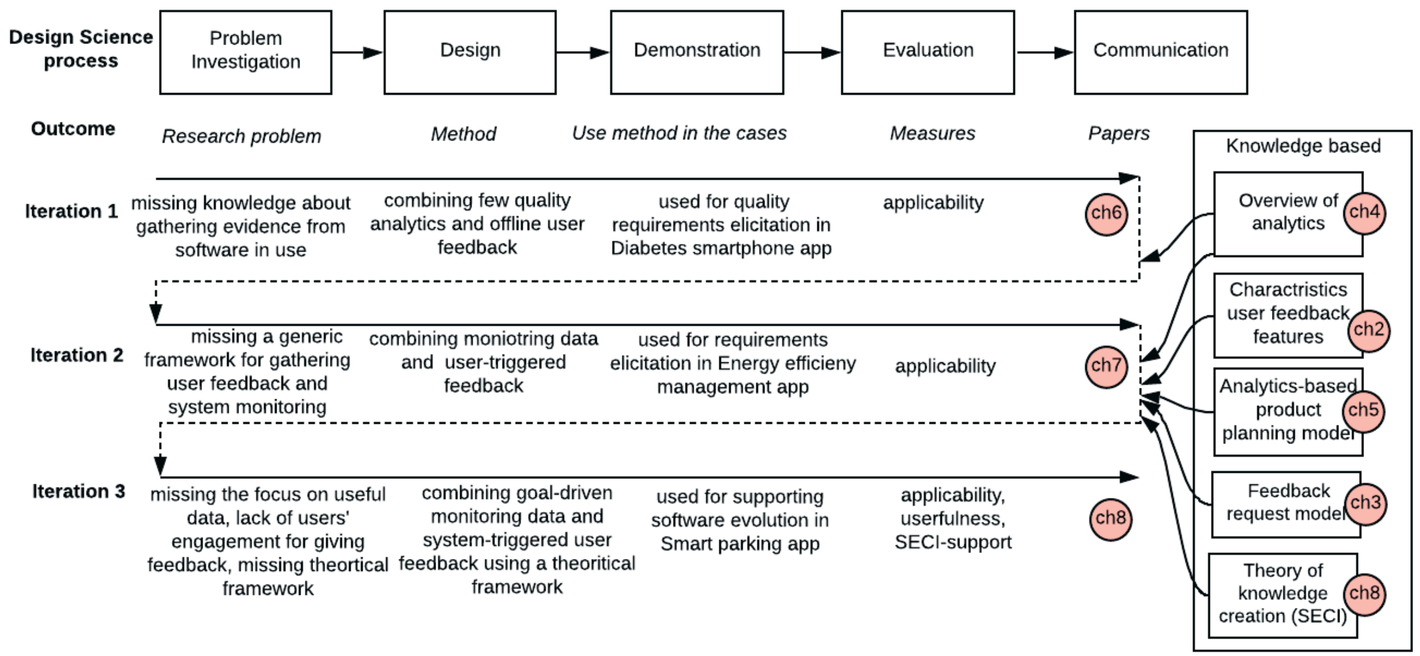

Design Science process
Problem Investigation
Design
Demonstration
Evaluation
Communication
Outcome
Research problem
Method
Use method in the cases
Measures
Papers
Iteration 1
missing knowledge about gathering evidence from software in use
combining few quality analytics and offline user feedback
used for quality requirements elicitation in Diabetes smartphone app
applicability
ch6
Iteration 2
missing a generic framework for gathering user feedback and system monitoring
combining moniotring data and user-triggered feedback
used for requirements elicitation in Energy efficieny management app
applicability
ch7
Iteration 3
missing the focus on useful data, lack of users' engagement for giving feedback, missing theortical framework
combining goal-driven monitoring data and system-triggered user feedback using a theoritical framework
used for supporting software evolution in Smart parking app
applicability, userfulness, SECI-support
ch8
Knowledge based
Overview of analytics
ch4
Charactristics user feedback features
ch2
Analytics-based product planning model
ch5
Feedback request model
ch3
Theory of knowledge creation (SECI)
ch8

Figure 1-2. Design science research method*

*: red circles refers to the chapters of this thesis where the knowledge discussed

Figure 1-2 provides an overview of the research methodology, which slightly adapted the design-science model proposed by Peffers et al. (2007). We replaced the term *demonstration* with *apply* to avoid terminological confusion, and we merged the process *define objective and solution* with the *design* process.

4.2. Case Study

The studies in Chapters 3, 7, and 8 used the case-study research method.

In **Chapter 3**, we evaluate how requests for user feedback about a software product affect the quality of experience of feedback senders. We recruited 35 software engineering students at the graduate level who were familiar with the concepts of requirement modelling. We sought a variation as large as possible among the participants and treated each student's product use as a case in a multiple embedded case study (Yin 2014).

For data collection, we assigned a requirement-modelling task to the students. The QoE probe feedback tool (Fotrousi and Fricker 2016) was used to request feedback randomly from participants while they were using the requirement-modelling tool Flexisketch (Wüest et al. 2015). At the end of the product's usage, we collected the students' perceptions of the feedback requests and the experiences of using the product through a post-questionnaire. To answer the research question, we analysed the user feedback from the software in use (i.e., feedback about software) and the user feedback from the post-questionnaire (i.e., feedback about software and the feedback request that the system triggered).

The study in Chapter 3 used a mixed qualitative-quantitative analysis. For the qualitative analysis, we chose inductive and deductive content-analysis approaches (Elo and Kyngäs 2008). The inductive approach was based on free coding data to generate information, and the deductive approach was based on the use of initial coding categories extracted from the hypothesis with the possibility of extending the codes (Hsieh and Shannon 2005). The study also used the pattern-matching analytical technique (Yin 2014) to test predicated patterns (hypothesis) in comparison with observed patterns. The study also performed a quantitative statistical analysis.

We conducted two single case studies (Yin 2014) in Chapters 7 and 8 to apply and evaluate GESU. We tested the technical applicability and usefulness of the method in a particular infrastructure, strengthened our theoretical understanding, and deepened our knowledge of the specific case (Ulriksen and Dadalauri 2016).

Chapter 7 investigated how combining user feedback and data monitoring could support continuous requirements elicitation. We used FAME (feedback acquisition and monitoring events) framework that we designed for the combined and simultaneous collection of feedback and monitoring of data in web and mobile contexts. We deployed FAME in the web application of a German small-to-medium-sized enterprise (SME) to collect user feedback and usage data.

To prepare the data-collection environment, we configured a feedback dialogue and included all the feedback mechanisms available in the FAME framework. We activated only a user-triggered mechanism: the users could trigger a feedback dialogue by pushing the feedback button available on every web page. We also configured the application to use only the monitoring tool relevant to usage to obtain the clickstream and navigation paths of end users.

We collected user feedback and monitoring data for four months. Afterwards, we conducted a small requirements elicitation workshop in two phases involving a researcher and an employee from the SME. In the first phase, the SME representative had to elicit requirements considering only feedback data as had been done to that point. In the second phase, he had to elicit further requirements or refine the previously elicited ones. For this purpose, the researcher provided the SME representative the relevant feedback entries identified in the previous workshop phase combined with the monitoring data. The combined data covered the time between user logins and sending the feedback, and it included the actions of the end user who provided the feedback and the list of end users who did not provide feedback but took the same actions. The representatives went through the combined feedback and monitoring data to further elicit new requirements or update available ones.

Chapter 8 proposes a method for GESU designed within the framework of knowledge-creation theory and proposes technical solutions to improve GESU. According to the proposal, the system monitors product goals to identify interesting situations of system use and issues automated requests for user feedback to gather evidence of software evolution from users' perspectives. We evaluated the method in a smart parking case study using observations of record and interviews. The case benefitted from the use of thousands of IoT traffic and parking sensors deployed in the city of Santander that helped users find free parking. We integrated our method with a recommender system that generated recommendations of unoccupied parking spots and pathways to them for end users.

Interviews combined with observation were the primary means of data collection. We ran a pilot study with citizens of Santander who volunteered because of intrinsic motivation to help the city's evolution as a smart city. The pilot study lasted three months, during which our method ran, and we gathered monitoring data and user feedback. After finishing the data collection, we synthesised the user feedback and monitoring data and then planned four interviews with three developers and one decision-maker in the case. During the interviews, we presented the results of the user feedback analysis to the interviewee, including the synthesised list of user feedback, the frequency of each feedback, and a map showing user feedback associated with sensor locations on the map. We asked questions regarding actions the interviewees would have taken with that knowledge. We sought to identify how the knowledge was transferred and shared, in what format, and whether anybody else was involved in this activity.

To analyse the interviews, we transcribed them and used a deductive content-analysis approach (Elo and Kyngäs 2008) to codify the transcripts. We then iteratively used explanation building (Yin 2014) to check the conformance of the interview data with the method.

4.3. Systematic Mapping Study

To learn about the analytics tools that product owners use, we conducted the systematic mapping presented in **Chapter 4** for an overview of analytics and KPIs. We chose the software ecosystem context, a broader area than a software system, to ensure that the analytics product owners used for relations between the systems were included in the search. The research thus provided an overview of the KPIs used in a software ecosystem by classifying relevant articles and mapping the frequencies of publications over corresponding categories to build classification schemas and observe the current state of research (Petersen et al. 2008). A systematic literature review is an alternative method, but it differs in goals and depth. The aim of the study was not to find the best practices based on empirical evidence; a broad overview was sufficient and preferable to the time-consuming process of sifting through details in greater depth.

We researched with the following four steps according to the guidelines introduced by (Petersen et al. 2008): searching databases, screening papers, building classification schemas, and systematically mapping each paper. In the database search, we defined the search string to include keywords relevant to the software ecosystem and KPIs. The search strings were entered into software engineering and computer science research databases, including Scopus, Inspec, and Compendex, which also includes IEEEXplore and the ACM Digital Library. In the screening step, we screened the identified papers to exclude studies unrelated to the use of KPIs for any ecosystem-related purpose. In the classification step, we employed keywording (Petersen et al. 2008) to build the classification scheme in a bottom-up manner. Extracted keywords were grouped under higher categories to make them more informative and reduce the number of similar categories. In the last step, when the classification was in place, we calculated the frequencies of publication for each category and used x-y scatter plots with bubbles at category intersections to visualise the generated map.

4.4. Survey Research

The studies in Chapters 2 and 5 used a survey research method.

In **Chapter 2**, we conducted a questionnaire-based survey to understand how feedback senders perceived the characteristics of a feedback feature in an embedded feedback channel in a software system, and how those characteristics influenced the use of the feedback channel. In the context of this research, we studied eight feedback features *text*, which are *rating*, *emoji*, *category*, *screenshot*, *audio recording*, *screen recording*, and *attachment.* We investigated *six characteristics of the features,* namely *perceived usefulness, perceived ease of use, familiarity, explaining a complex situation, tailoring* and *transferring emotions*.

We designed a questionnaire based on a conceptual model that we built on theoretical priors. We implemented the questionnaire and collected answers using QuestionPro. A known public panel (i.e., consumerfieldwork.com) helped us recruit respondents for 100 completed questionnaires.

In the questionnaire, we asked about the *familiarity* of the respondents with giving feedback. We introduced an exemplary embedded feedback tool to ensure shared, equal understanding of the feedback features descriptions. We asked the respondents to evaluate whether the example feedback form in total, and each feedback feature separately was *easy to use* and *useful* and whether they were *familiar* with the feature and would intend to *use* it. We also asked the respondents to evaluate the capabilities of the feedback features to *explain a complicated situation*, *transfer emotions*, and *tailor* each feedback feature separately. Meanwhile, in several open-ended questions, we asked the respondents about the reasons for their choices. We also designed three scenarios that explained situations (e.g., bugs, new-feature requirement requests, and feature improvements) where the subjects chose their preferred feedback features.

We analysed the answers both quantitatively and qualitatively. We used descriptive analysis (e.g., Min, Max, Mean) to statistically explain the data, and significant analysis test (i.e., Krushkal-Walis H, independent t-test) to evaluate the significant differences between the groups of independent variables (e.g., age-group, education-group, and triggering approaches). We also used structural equation modelling analysis with a partial least-square algorithm (PLS-SEM) to find the structural relationships between the multivariable investigated statistically. We used the SmartPLS software version 3.2.9 for this purpose. For qualitative analysis, we performed an inductive content-analysis approach (Elo and Kyngäs 2008) to analyse and code the qualitative data provided in the questionnaire.

For **Chapter 5**, we conducted an interview-based survey to identify the relationship between analytics and product-planning decisions. We performed data collection using semi-structured phone interviews and asked a well-established consultancy company in software product management to introduce product managers experienced with software as a service (SaaS). SaaS products have the advantage of running applications in a cloud and could thus benefit embedded monitoring systems. We selected 17 product managers from three micro, four small, seven medium, and three large companies and conducted online semi-structured interviews with them. The interviews were piloted by two product managers and two students with product-planning knowledge. After initial testing and several refinements, the interviews with the product managers were scheduled.

Questions about product planning formed the core of the interviews in two parts: planning decisions and analytics. In the first set of questions, the interviewees were asked to select a product that they had planned and were most satisfied with. Then questions were asked about the planning decisions that the interviewees had made for the selected product. Later, the interviewees were asked to rate the importance levels of measurement categories and measurement attributes for making those decisions and then to comment on their reasons for those selections.

We used an inductive content-analysis approach (Elo and Kyngäs 2008) to analyse and code the interviews. We recorded the interviews and transcribed them for coding. During the analysis, we iteratively tagged units of arguments with headings, grouped the headings, categorised them, and performed abstraction. The process led to a model for analytics-based product planning and further discussions about the findings.

4.5. Descriptive Evaluation

We evaluated GESU in its first iteration presented in **Chapter 6** using a descriptive method (Hevner et al. 2004). A descriptive approach that Hevner et al. mentioned is constructing detailed scenarios around an artefact to demonstrate its utility. Our evaluation was also based on sample scenarios and provided arguments as proofs-of-concept. Wieringa et al. (2006) also considered such an evaluation method as part of a solution-proposal research category.

The novel solution proposed in Chapter 6 had the form of a technical approach that combined user feedback and monitoring data. We applied the solution (i.e., the method) to a diabetes smartphone application. To provide example data, we organized an inquiry workshop with representative stakeholders, including a requirements engineer, a product manager and a selected end-user. We used a prototype with software for monitoring the timing of user interactions and used a questionnaire to collect user perceptions. We finally explained through an example how the method's activities worked.

5. Chapters Overview

5.1. Summary of Results

We developed a method for GESU for software evolution that combines system monitoring with user-feedback collection and surface knowledge of the system subject to maintenance and evolution. As mentioned earlier, GESU was designed in three iterations in which various approaches to monitoring system data and collecting user feedback were applied. Chapters 6–8 elaborate on each iteration, which used the knowledge gathered from the studies in Chapters 2–5.

Figure 1-3 demonstrates the third iteration of GESU as presented in Chapter 8. GESU-3 visualises two dimensions: knowledge transformation and its activities (answer to RQ3). The method addressed eight activities for evidence-based software evolution: *preparing and running a software system* (Activity 1), *monitoring system and usage* (Activity 2), *collecting user feedback* (Activity 3), *aggregating monitoring data and user feedback* (Activity 4), *collecting product team feedback* (Activity 5), *socializing* (Activity 6), *aggregating previous evidence with product team feedback* (Activity 7), and *implementing changes* (Activity 8). Since the first four activities were more data-intensive than the others, their automation would facilitate the evidence-gathering. Therefore, in all three iterations of GESU, the four activities were the focal points.

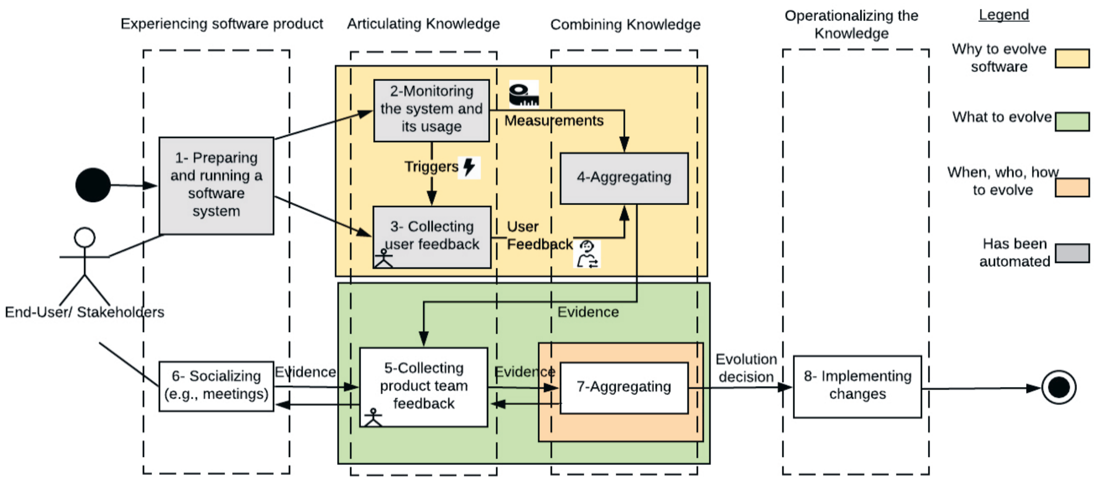


Figure 1-3. Overview of gathering evidence from software in use (GESU)

Table 1-2 summarises and compares Activities 1–4 of GESU along with its three iterations (GESU-1 to GESU-3). In the first iteration (GESU-1) presented in Chapter 6, we aimed to investigate the feasibility of combining evidence using the common requirements-elicitation methods of user feedback and monitoring data. We integrated the QoE probe with a diabetes mobile application (Activity 1) to monitor three quality attributes from the software in use (Activity 2) and collect offline feedback through a questionnaire (Activity 3). Using correlation analysis, we aggregated the data (Activity 4) to help gather evidence to estimate the proper value of corresponding quality requirements.

In the second iteration (GESU-2) presented in Chapter 7, we aimed to develop and use a generic framework to improve gathering and combining user feedback and monitoring data. We implemented FAME and integrated it with an energy-management application. We set up the framework (Activity 1) to monitor usage (Activity 2) and gathered user feedback with user-triggered feedback (i.e., the users pressed a feedback button to give feedback) (Activity 3). We then aggregated the monitoring data and user feedback using descriptive graphs that could help requirements engineers gather evidence for new requirements or update available ones.

In the third iteration (GESU-3) presented in Chapter 8, we aimed to improve gathering evidence. GESU-3 introduced goal-driven monitoring and system-triggered user feedback. We implemented this goal-driven monitoring approach in a smart-city application, configured FAME, and integrated it with the application to gather user feedback (Activity 1). We then monitored the goals defined for the application (Activity 2). When the measurements corresponding to the goals deviated from their accepted threshold values, the system automatically configured and triggered a feedback request to gather evidence about the deviation (Activity 3). Using goal-driven monitoring allows product teams to focus on those measurements, which facilitates software evolution. GESU-3 could complement the common approach of system monitoring and gathering user-triggered feedback to find evidence for software evolution. We also integrated monitoring data and user feedback using a visualisation map (Activity 4), a tool that could reveal new evidence for evolving software systems.

Table 1-2. Evaluation of GESU in three iterations

Method	**Preparing (Activity 1)**	**Monitoring (Activity 2)**	**Collecting user feedback (Activity 3)**	**Aggregating (Activity 4)**	**Chapter**
GESU-1	Set up QoE probe, integrate it with a diabetes app, and prepare offline feedback forms	Limited (three) quality attributes monitoring	Offline feedback at the end of usage	Correlation analysis	6
GESU-2	Set up FAME and integrate it with an energy-management app	Usage monitoring	User-triggered feedback	Descriptive graphs	7
GESU-3	Set up monitoring app and FAME user feedback and integrate the tools with the smart-city app	Goal-driven monitoring + system monitoring	System-triggered feedback + user-triggered feedback	Visualisation map	8

We use goal-driven and system-triggered feedback in GESU-3, as suggested by other findings described below.

Chapter 4 overviews and classifies the goals in a software ecosystem and relates them to KPIs. This classification contributes to taxonomy, which can help closer examination of the ecosystem or platform owners' goals, making them more recognisable in designing the goals. As Chapter 5 shows, these goals can help decision makers such as product managers focus on observing particular analytics instead of collecting massive amounts of useless data.

In Chapter 2, we showed that perceived *usefulness*, *ease of use* and *familiarity* (with feedback features and giving feedback) are the most influential factors on the use of a feedback channel. And, the capabilities for *explaining a complicated situation*, *tailoring and transferring emotions* have the least impact on the use. We summarised the strengths and limitations of feedback features of an embedded feedback channel. The study could confirm text is the most preferred feedback feature that is perceived as *useful* and *easy*, and many users are *familiar* with that. However, there are other feedback features such as *screenshot*, *and audio recording* that have more influential contributions on *explaining a complicated situation* and *tailoring* the feedback to the needs. The study suggests considering both user-triggering and system-triggering approaches for gathering user feedback. However, the requests for user feedback in a system-triggering approach should not disturb users, as shown in Chapter 3, although the disturbance has a negligible impact on quality of experience. Having the two approaches enable the users who have the habit of giving feedback to give the feedback when intend, and motivates the users, who do not have the inclination, to communicate their input.

5.2. Summary of Chapters

Chapter 2 presents an exploratory study that investigated the characteristics of feedback features of an embedded feedback channel from the perspectives of feedback senders. It also studied how the feedback characteristics impact the use of the feedback channel. We chose six factors for evaluating feedback features by inspiration from technology acceptance model and media richness theory, namely *usefulness, ease of use, familiarity* level of users with the feedback feature and giving feedback in addition to capabilities for *explaining a complicated situation, tailoring feedback*, and *transferring emotions*. In total, we selected eight feedback features including *text, rating, emoji, category, screenshot, audio recording, screen recording,* and *attachment*. The respondents of a questionnaire-based survey in the role of feedback senders evaluated the features for the chosen characteristics.

The results confirmed that perceived *usefulness, ease of use,* and *familiarity* (e.g., with giving feedback, and with feedback features) are essential factors affecting the *use* of a feedback channel. However, the study could not find strong evidence that feedback capabilities for *explaining a complex situation*, *tailoring*, and *transferring emotions* for a significant impact on a *use* of a feedback channel (answer to RQ1.2). *Text, rating, emoji, category* and *attachment* were perceived to be *easy to use* features while user was *familiar* more with *text, rating* and *category*. Those feedback features which are perceived to be *easy to use*, are not good in *explaining a complicated situation*. Although qualitative analysis of the quotes of feedback senders showed that the study could confirm *text* is the most preferred feedback feature, however, there are other feedback features such as *screenshot*, and *audio recording* that have more influential contributions on *explaining a complicated situation* and *tailoring* the feedback to the needs. The study could not find significant differences across categories of age for the preferences of feedback features (answer to RQ1.1).

Overall, the study suggests providing feedback forms with multiple feedback features to allow users when they push the feedback button to be able to choose among the features based on their preferences and needs. As a complementary approach, we also suggest companies to design simple feedback forms (with one or two feedback features such as *text* and *rating*), customized based on the situations that the users just experienced, and then use the form to request users for feedback. We expect that this approach would engage those users that do not have a habit of giving feedback. The findings imply that companies need to consider courses of action (e.g., tutorials) to make their users familiar with the feedback features, engage them to give feedback (answer to RQ1).

Chapter 3 presents empirical research that explores how requests for feedback affect users' QoE. In this study, a QoE probe feedback tool (Fotrousi and Fricker 2016) was integrated with a mobile product. The feedback tool randomly prompted users for feedback about the product and collected users' perceptions of their experiences during their interactions with the product. At the end of the experience, a post-questionnaire collected users' feedback about the feedback tool and the software product.

The analysis of the users' feedback about the feedback tool identified categories of user disturbance. Users perceived disturbance when feedback requests interrupted their tasks, when they were too early, too frequent, or had unsuitable content. These findings helped parameterise the characteristics of feedback requests and a *feedback-request model* with four parameters referring to the *experience space, the time frame within the space, the number of feedback requests in the time frame, and the content of the feedback* request. This model implies that user disturbance may be avoided by a suitable configuration of these variables.

The analysis of the users' feedback about the software product showed that the disturbances generated by the feedback tool had a negligible impact on the QoE of the software product, however (answer to RQ1.3). Triangulating the study with three different analyses confirmed the finding: a pattern-matching analysis showed that the disturbance caused by the feedback tool did not always create a bad experience of the software product. Correlation analysis confirmed that the QoE of the software product was not statistically correlated with the QoE of the feedback tool. The content analysis of the users' feedback showed that the QoE of the software product was essentially influenced by other factors. The quality of the software product and the context of the user experience, such as device characteristics, were the focal points of users' justifications for their ratings.

The negligible impact of feedback requests on QoE implies that software practitioners may trust the evidence that their feedback tools collect even if they disturb users. The feedback can be informative about software products or even about the disturbing feedback tool itself. These findings can guide designing system requests for user feedback to enhance the evidence they gather.

The results of the study were limited to the experiences of students with the modelling requirements of a mobile software application. The contextual factors might also have affected the results. In the future, other studies with different types of products should complement the current study to confirm the reliability of its results. Another open question exists regarding how variations of the feedback-request parameters affect QoE.

Chapter 4 gives an overview of the literature on the use of KPIs, which target product goals, for software-based ecosystems. A systematic mapping was followed and applied to 34 studies published from 2004 onwards.

Two major kinds of ecosystems were researched: *software ecosystems* and *digital ecosystems*. Many application domains, such as *software development*, *telecom*, *business management*, *logistics*, *transportation*, and healthcare, were addressed, but most in only one or two studies.

The mapping study showed that KPIs relate to a variety of objectives: improving business, the interconnectedness between actors, ecosystems, the quality of ecosystems, products and services within ecosystems, and ecosystem sustainability.

The included primary studies described KPIs applied to whole ecosystems or parts of ecosystems, which consist of entities including *actors, artefacts, services, relationships, transactions*, and *networks.* They were identified in relation to the ecosystem objectives. To measure the entities, we classified the KPIs into the categories *size*, *diversity*, *satisfaction*, *performance*, *finance*, *freedom from risk*, *compatibility*, and *maintainability* (answer to RQ2.1).

Among the primary studies, the most common objectives were improving the interconnectedness between individual actors, ecosystem subsystems, and the quality or business of the overall ecosystem. Satisfaction, performance, and freedom from risk were the most common KPI categories.

The classification provided in this study can help ecosystem, platform owners, and even software system managers to more closely examine objectives and design relevant to KPIs (partially answer RQ2). The results of the mapping study indicate that more research is needed for a better understanding of KPIs for software-based ecosystems. In particular, a deeper understanding is needed of how the application domain affects an ecosystem's KPIs. Further research on the identification and analysis of KPIs can improve the understanding of which KPIs are best suited for which purposes.

Chapter 5 describes whether and when analytics are valuable for product planning and how they can be used in a software product plan. The chapter reviews the existing literature on software product planning and analytics and proposes a conceptual model that connects software product analytics to product-planning decisions.

This study initially introduced two taxonomies as inputs for the other parts of the study: the first was related to planning decisions taken in portfolio management, road-mapping, and release-planning. The second, which was a taxonomy for SaaS-based measurements, had two dimensions: *product*, *feature/content*, and *GUI (graphical user interface) elements* in the first dimension and *health*, *usage*, and *context* in the second.

We conducted an interview-based survey with experienced product managers focussed on road-mapping decisions to show how analytics assist product managers in product planning. We asked the product managers to rank the analytics they used for their decision-making. We then analysed their rankings and rationales.

The results presented an analytics-based model (answer to RQ2.2) that indicates that both product characteristics and product goals constrain analytics and that analytics can be interpreted for product goals. The study also suggested activities for product managers to support planning decisions and product evolution with analytics: extracting product characteristics and preparing a list of product goals (Step 1), filtering lists of analytics based on product characteristics and goals (Step 2), measuring and analysing (Step 3), and making planning decisions while checking alignment with goals (Step 4).

The results also revealed that some parameters, such as product maturity, users, network type, context, and technology, can change the value of analytics for product planning. Analytics can be motivators for product managers to achieve goals for market positioning, meeting quality-in-use, and improving product quality (e.g., usability, functional suitability, and reliability). Therefore, even a limited list of analytics supports decision-making based on actual evidence. If analytics show deviations from product goals, the product manager can make an informed decision (partially answer RQ2).

Chapter 6 describes an approach to eliciting quality requirements based on inquiry into quality-impact relationships. The method is based on the quality of a product and subjective feedback from stakeholders about perceived quality to guide a requirements engineer in the systematic inquiry of good-enough software quality. Chapter 6 is a solution proposal that describes the method in detail and reports early experiences of applying it.

The proposed method is performed in four steps: *preparation*, *measurement*, *analysis*, and *decision-making*. During *preparation*, the required materials are prepared, including the preparation and documentation of a prototype, the formulation of a questionnaire, the recruitment of stakeholders for participation in a workshop, and the scheduling of the workshop. The *measurement* step aims to collect quality measurements and user feedback. While stakeholders use the software, the quality attributes of the experience are measured. After using the software, a questionnaire is administered to collect stakeholder opinions about perceived quality. In the *analysis* step, quality measurements are correlated with stakeholder opinions. The analysis uses a regression function to estimate what stakeholder opinions would be for a given quality value. During the *decision-making* step, the requirements engineer decides the acceptable and desired levels of quality for the investigated quality attributes and updates the SRS (software requirements specification) document if needed (answer partially to RQ3.2). The process concludes with decision-making about whether to add inquiry iterations.

We applied the method to a real-world project in the healthcare domain and exemplified the steps using a diabetes smartphone application. Diabetes patients used the product to take blood glucose measurements, plan insulin injection, and send the collected observation history to a diabetes specialist for a consultation. The relationship between quantitative measures of user opinions and system measurements was evaluated for the features of authenticating user and sharing diabetes information. Chapter 6 describes how the method was applied to the requirements engineering endeavours. The method combined evidence from the elicitation methods, particularly from questionnaires, monitoring prototypes, and workshops, into a structured process for creating and analysing evidence for decision-making about good-enough quality.

Chapter 7 presents the FAME framework, which was designed and implemented for the combined and simultaneous collection of feedback and monitoring data in web and mobile contexts. FAME gathers user feedback, monitors data, and stores data. FAME uses an ontology that links user feedback and monitoring data via their shared schema elements, including user, timestamp, and application, to help practitioners find evidence for evolving software systems.

FAME supports several feedback media, including *text, rating, screenshot, audio, category, and attachment*. It also supports both user-triggering (push) and system-triggering (pull) approaches to feedback dialogues and implements several monitoring approaches: user monitoring (e.g., clickstreams and navigation paths of end users), infrastructure monitoring (e.g., disk, memory, and central processing unit [CPU] consumption of a server), and QoS monitoring (e.g., monitor response time and availability of web services). The implementation is in the form of a library that a developer integrates into a host application. Gathering user feedback and monitoring data are fully configurable, allowing feedback dialogues to be created and monitors selected based on the stored configuration settings.

We evaluated FAME in the context of continuous requirements elicitation. FAME was deployed in the web application of a German SME to collect user feedback and usage data. We received 31 feedback entries from 24 end users, and 16 were considered relevant. FAME also gathered monitoring data from all 5,185 end users who logged in during the period, 957,260 clicks, and 160,888 navigation actions. After the combined analysis of user feedback and monitoring data with the SME representative, we found evidence for one new requirement and four modified requirements that the representative had not mentioned earlier.

These results suggest that FAME is not only successful in industrial environments but also that bringing feedback and monitoring data together helps company stakeholders improve their understanding of end-user needs, supporting continuous requirements elicitation and ultimately software system evolution (answer to RQ3.2). As future work, FAME should be validated with companies of different domains, sizes, and requirements elicitation processes as well as with more end users.

Chapter 8 presents GESU in detail (answer to RQ3). GESU explains two dimensions: knowledge transformation (by adapting the SECI model) and its activities, the details of which were presented in Section 5.1.

We evaluated GESU in a smart parking case, the results of which showed that GESU could effectively support decision-making for software evolution and that combining user feedback, monitoring data, and the product team's knowledge could support software system evolution (answer to RQ3.2). The findings also confirmed that GESU could describe knowledge-creation theory, although it needed some adaptation for the specific case (answer to RQ3.3). Additionally, the results showed that goal-driven monitoring and system-triggered user feedback approaches could yield more contextual evidence and consequently help improve gathering evidence (answer to RQ3.1). For further contributions, the paper revisited the SECI model as the conceptual basis of gathering knowledge from software in use in the context of system maintenance and evolution.

6. Discussion

6.1. Contributions

This dissertation makes the following eight contributions as follows. For each contribution we identifies which objectives (see Section 3) have been fulfilled.

C1: Describing a method that guides practitioners and researchers with generic activities and their sequence defined. The method is framed based on processes for knowledge creation and transformation to support gathering and sharing evidence to evolve software systems. The method called GESU (gathering evidence from software in use) in this thesis (OBJ 3, OBJ4, OBJ4.2, OBJ 4.3).

C2: A better understanding of goal-driven monitoring of a system in use for requesting user feedback relevant to the use context. A goal-driven approach guides monitoring to observe particular analytics according to the goals. When the system detects a deviation of analytics from a threshold value, the system configures and triggers a request for user feedback (OBJ 3, OBJ 4.1, OBJ4.2).

C3: A better understanding of gathering user feedback from software in use. We specified the characteristics of various feedback features of a feedback channel embedded in a software system, and the strengths and limitations of the feedback features. *Perceived usefulness, perceived ease of use*, and *familiarity* with the feedback form would improve the use of feedback channel. The embedded feedback channel, which contains several features (e.g., free text input, star rating, and taking and annotating a screenshot), is useful for collecting more information. However, requesting feedback in a simple feedback form that is relevant to the context of the users' recent experience encourages more users to give feedback. The findings contribute to helping practitioners and researchers to design an embedded feedback channel while improving its use (OBJ1, OBJ1.1, OBJ1.2, OBJ 1.3).

C4: A better understanding of the effect of requests for user feedback on QoE. We found that feedback requests have a negligible impact on users' QoE of the software product as such. The quality of software products themselves has much more impact on QoE than the characteristics of the feedback tool, implying the importance of practitioners' focus on product quality, although designing a proper feedback tool should not be neglected, since it contributes to collecting informative feedback about the software product. These findings contribute to helping researchers and practitioners design automatic user-feedback request systems (OBJ1, OBJ1.4)

C5: A better understanding and modelling of the characteristics of a feedback request that may disturb users. The study parameterised the characteristics of feedback requests to inform researchers about the factors that disrupt users' experiences and thus help them design feedback mechanisms that avoid disturbing users (OBJ1, OBJ1.4)

C6: Implementation of the FAME framework and QoE probe tool. FAME provided a framework with fully configurable tools for collecting user feedback, monitoring data, and combining the two. The configurable user feedback tool was implemented in both the web and Android versions. The Android mobile QoE probe tool collected limited monitoring data as well as quantitative and qualitative user feedback. These tools can be used to develop feedback-based research projects and evaluate software products based on users' feedback (OBJ1, OBJ2, OBJ3)

C7: A better understanding of platform owners' objectives and relevant KPIs that are measured, analysed, and used for decision-making in a software ecosystem. From a researcher's point of view, the study captures the state of knowledge and can be used to plan further research. From a practitioner's view, the generated map refers to studies that describe how to use KPIs to manage a software ecosystem (OBJ2, OBJ2.1).

C8: A better understanding of how analytics can be used for product planning. We found that analytics are used to interpret product goals, while analytics are constrained by both product characteristics and product goals (OBJ2, OBJ2.2).

6.2. Roadmaps

The thesis theoretically and practically targets gathering, sharing and aggregating evidence for software evolution in a systematic way. We proposed a theoretical framework of knowledge creation for companies to not only consider gathering evidence but also take actions for combining and sharing evidence for software evolution among their team. However, the thesis practically focused on the activities relevant to gathering evidence from monitoring data and user feedback, and combining them due to the data-intensive nature of these activities, which facilitates their automation.

Although many companies are aware of the importance of user feedback, they are not aware of a suitable approach for gathering feedback and exploiting the feedback potentials (Stade et al. 2017). We believe that a success factor for the use of feedback forms by users is to match the capabilities of a feedback form to the users' needs for communicating their thought. We suggest companies design feedback forms while including multiple feedback features such as *text* feedback (popular feedback), *rating* (easy to use), *audio recording* (capable of explaining a complex situation), allows integrating diverse characteristics of the features. The feedback forms should be available for users to trigger, for example, by pushing the feedback button, when they have feedback to communicate. Furthermore, we suggest designing simple feedback forms, including one or two easy to use feedback features, such as text and rating, adjusted to the context of the recent experience of the users. The system triggers the feedback form to request feedback from the users. However, such system requests for feedback should not disturb the users (see Chapter 3). Combining the two approaches enable the users who have the habit, to give feedback when intend, and also engage other users, who do not have the inclination, to communicate their input just after experiencing it. However, for designing feedback forms, the researchers and practitioners need to consider ethical aspects and ensure that the stakeholders' rights and integrity are respected (Fotrousi et al. 2017).

We suggested companies considering goal-driven monitoring and combining the monitoring with user feedback. The approach steers the collection of user feedback on interesting situations of system use by basing the feedback requests on monitoring the fulfilment of user goals. This method avoids collecting volumes of unused data and reduces the number of disturbances of users due to feedback requests by collecting short feedback only when it is needed. However, unnecessary disturbances can also negatively impact the amount of feedback received, even when it has a negligible impact on the quality of the experience (Fotrousi et al. 2018). This supports practitioners and managers in gathering actionable evidence from a system and its users to support the iterative approaches of continuous product development and delivery.

The method proposed in this thesis is particularly useful for continuous deployment and delivery (DevOps) as it can bring software development closer to operation (Sjøberg et al. 2003). Using the method, the practitioners act on evidence from system monitoring and user feedback to mitigate issues as quickly as possible. So, most users are no longer affected by the issues, and the practitioners can increase the frequency and quality of deployments while making the development lifecycle shorter. Developers can also cut work into small chunks and carefully iterate the flow of products to users, as followed in lean start-ups (Blank 2013).

The method proposed in this research applies to software-intensive systems, such as IoT (internet of things) (Sanchez et al. 2014) and cyber-physical systems (Zheng et al. 2016). These systems are usually characterised as spatially distributed hardware and software, and time-sensitive, connecting the physical world to the cyber world through sensors and actuators (Esterle and Grosu 2016). The method in this thesis can be used by introducing the concept of goal-driven to individual monitoring of the hardware and software components, aggregating the monitoring data, and combining them with autonomous user feedback. Such a system can benefit from a self-adaptive user-feedback to automatically detect problems and provide a feedback loop for automatic fixes. This self-adaptive user feedback has been left for future research.

We can apply our method in various other disciplines such as in transport, energy, healthcare, and agriculture domains. The method proposed in this thesis enables transferring knowledge from a laboratory and testing environment to a real environment (e.g., roads, hospitals, and farms), and open up opportunities for future research projects.

6.3. Limitations

This thesis does not aim to prescribe a solution for gathering evidence in a particular context. The thesis provides a procedure that, on the one hand, frames the processes relevant to GESU and, on the other, recommends goal-driven system monitoring and system-triggered user-feedback gathering. This is alongside typical methods, such as continuously monitoring whole systems and gathering user feedback when users trigger feedback forms. Goal-driven monitoring is applicable only if software goals and corresponding KPIs (i.e., monitor analytics toward a goal) are defined; otherwise, practitioners should rely on typical system monitoring.

GESU method suggests goal-driven user feedback gathering, which can rely on a variety of monitoring techniques and algorithms, for example, a techniques for managing the traffic overheads of monitoring in distributed system (Dilman and Raz 2002). However, the proposed method is agnostic about those techniques and algorithms and does not elaborate on them.

Gathering evidence requires tools to monitor systems and collect user feedback. The current version of FAME does not support goal-driven monitoring, so practitioners should either wait for an update of FAME or act on customising the current solution based on their particular needs.

6.4. Future Work

Building an adaptive user-feedback system would be a valuable future work. Such a system relies on parametric self-configuration of user-feedback forms. The system triggers a request for user feedback at an appropriate time while the user is experiencing the software. The model presented in chapter 3 parameterises a feedback request outline the task, timing of the task for issuing the feedback requests, user's expertise-phase with the product, the frequency of feedback requests about the task, and the content of the feedback request. Content could refer to the selection of appropriate user-feedback features (see Chapter 2) with questions adjusted, for example, based on user behaviour in the running software and prior user feedback. So, monitoring data could also form other parameters. How to parameterise a user feedback system, and how to make it self-configurable could be the subject of future research.

An extension to the FAME framework (see Chapter 7) should be developed that, on the one hand, supports goal-driven monitoring and, on the other, makes the self-adaptive solution practical by configuring (self)-adaptive user-feedback forms in a controlled loop. A validation of the extension should also be planned.

A systematic approach to designing and executing innovation experiments in iterative software development, such as in lean start-ups, is also interesting future research. The method for proposed for gathering evidence in this thesis can inform problem detection and guide code changes to speed up production. What is still unknown in the literature is how to formulate hypotheses regarding software products and prioritise hypotheses using the collected evidence. We think that combining monitoring data and adaptive user feedback with users' own reasoning about the causes of unfulfilled expectations could be a solution to the formulation of system hypotheses.

When relevant hypotheses are formulated and prioritised, experimentation requires building prototypes for the hypotheses with *the minimum viable product*, observing user behaviours and feedback, and perhaps repeating the process with a new hypothesis. Once enough support is found for a hypothesis, the product is developed and released. Due to various conceptual factors and the manual effort of building prototypes, a solution for automatic experimentation does not seem feasible. However, research is needed to instruct practitioners about approaches to combining user feedback and monitoring data to support experiments.

In such studies, trust plays an important role, especially when innovative autonomous demonstrators are introduced, such as self-driving software. It would be interesting to investigate the impact of new software or its changes on human trust. The proposed method in this thesis should be helpful for measuring trust, in fact. Whether and how trust can be measured using user feedback and monitoring systems should also be researched in the future.

7. Conclusion

Companies must acquire evidence about situations in which their running software systems misbehave and generate undesirable impacts that users do not accept. Previous research has collected user feedback and monitoring data as two sources of evidence; however, the gathering of such evidence has not been systematic, resulting in overreliance on the passive reactions of users when a problem emerges and finding issues among massive amounts of monitoring data. This thesis aimed to improve this evidence-gathering from user feedback and monitoring data to support practitioners in decisions regarding the evolution of software systems.

The thesis proposed GESU using a design-science research method designed in three iterations and validated with the case studies of the European projects FI-Start, Supersede, and Wise-IoT. GESU was built on the results from other supportive studies of this thesis. We found that users characterise feedback features differently. Although some users have the habit of giving feedback, some others will communicate their feedback if the system solicits particular information about recent experiences. However, requests for user feedback should be infrequent or only occur after finishing feature usage to avoid disturbing users, even though the disturbance may have a negligible impact on the perceived QoE of the software system. We modelled analytics-based planning that showed analytics could support the interpretation of product goals but are constrained by both product characteristics and product goals.

GESU was built within the framework of knowledge-creation theory and suggested goal-driven monitoring and system-triggered user feedback for gathering evidence from software in use. The evidence-gathering relies on choosing analytics based on product characteristics and product goals, monitoring analytics, and requesting user feedback when the monitoring data shows deviations from accepted threshold values. Adding a joint analysis and presentation of the monitoring data and user feedback is recommended to discover evidence of problems or required changes.

The results show that GESU is not only successful in industrial environments but that bringing feedback and monitoring data together helps practitioners improve their understanding of end-user needs and system drawbacks, ultimately supporting continuous requirements elicitation and product evolution.

Overall, combining user feedback and monitoring data is helpful to deepen insight into the success of software systems and guide decision-making regarding software maintenance and evolution. This work can be extended in the future to incorporate an adaptive system for gathering evidence from combined monitoring data and user feedback. Such an adaptive system may guide hypothesis formulation and testing to further bolster continuous system maintenance and evolution

PART 2

Gathering of User Feedback

Chapter 2 : How do Users Characterise Feedback Features of an Embedded Feedback Channel?

[The chapter is based on: ***F. Fotrousi****, M. Stade, N Seyff, S. Fricker, M. Fiedler (2020). "How do Users Charactrise Feedback Features of an Embedded Feedback Channel?", Submitted to a Journal*.]

Abstract

Feedback from users is a valuable source of information for software evolution. The evolution can be informed using a feedback channel embedded in the software, to allow users communicating their desires, needs and experienced issues at runtime in the forms of text descriptions, annotated screenshots, ratings and even audio recordings. Different users have different needs when providing feedback. Although the provision of a suitable feedback channel for users is crucial to let them communicate their feedback as they need, little is known under which conditions users prefer to use the different available feedback features. Our study aims at understanding the characteristics of a feedback feature of an embedded feedback channel such as ease of use or transferring emotions, and whether the characteristics have an impact on the use of the feedback channel. A survey was conducted with 100 public respondents. In total, eight feedback features were evaluated by feedback senders with regard to the traits inspired by the media richness theory and the technology acceptance model. The results confirmed that perceived usefulness, ease of use, and familiarity (e.g., with giving feedback, and with feedback features) are essential characteristics affecting the use of the feedback channel. The study showed that natural language *text* is the most preferred feedback feature, however there are other feedback features such as *screenshot*, and *audio recording* that have more influential contributions on explaining *a complicated situation* and *tailoring* or personalizing the feedback as they wish. Overall, the study suggests that providing feedback forms with multiple feedback features selectable by users based on their preferences and needs might be an ideal solution, when users trigger feedback forms. It also suggests providing a complimentary, simple feedback form to actively request feedback from users, for example, about the situations that the users just experienced. Asking for such immediate feedback could be triggered by the software system itself. Considering both triggering approaches allow software providers to keep users engaged in providing feedback regardless of whether they have a habit of providing feedback or not.

Keywords

User Feedback, Software Evolution, Feedback Features, Embedded Feedback Channel

1. Introduction

Five decades ago (Lehman 1980) emphasized that the evolution of software is *feedback driven* because feedback, in general, promotes change processes. In particular, feedback from users (i.e., end-users) is an impetus for change (Godfrey and German 2008) and a valuable source for software evolution. Software companies learn about the real software usage and receive improvement ideas (Maalej and Pagano 2011) as user feedback can cover various topics including shortcomings, feature requests or bug reports (Pagano and Maalej 2013). Such learning can make the products closer to the users' desires and needs, address the competition pressures, and generate values for software companies (Lehman and Ramil 2003; Thew and Sutcliffe 2018).

In order to benefit from user feedback for software evolution, user feedback needs to be properly managed in the multi-stage communication process between users and software providers (Gallivan and Keil 2003). The provision of suitable channels for users to communicate their feedback is a crucial step towards successful user-software provider communication. While some companies rely on generic communication channels such as phone, email or social media (Pagano and Brügge 2013; Stade et al. 2017) more and more companies integrate dedicated feedback tools into the actual software application. Such in-product implementations are seen as efficient means for users to provide feedback, incidentally while using the software. It can be argued that this increases the users' willingness to provide feedback.

The existing embedded feedback channels, sometimes referred to as feedback plug-ins or feedback tools, ranging from simple forms with free text fields, over pop-up windows with star rating requests to advanced screenshot-based plug-ins with manifold annotations options. Service providers of such feedback tools are competing with each other by providing (supposedly needed) new functionalities, up-to-date designs and the different feedback features supported. Feedback features belong to a feedback channel (also called *elements*, cf. (Schneider 2011)) and enable users to provide their feedback in different ways. This includes natural language text or spoken language, screen recordings, and emojis or star ratings. These different feedback features can be used to document and communicate feedback from the user (i.e., the feedback sender) to the software provider (i.e., the feedback receiver).

While current research focuses on the analysis of user feedback to support the feedback receiver side, for example, by applying machine learning algorithms to automatically classify feedback into categories relevant for software evolution (Guzman et al. 2015), the feedback sender perspective is often neglected. So far, we only know that users have different needs when providing feedback in general, (Almaliki et al. 2014; Maalej et al. 2009) but it is unknown under which conditions and for what reason users prefer to use a particular feedback feature to communicate their feedback (Maalej et al. 2009).

In this paper, we investigate how users rate and select the feedback features provided in embedded feedback tools to communicate their feedback. We focus on the feedback features available in research and industrial feedback tools, including text, screenshot, emojis and further features. In total, eight feedback features were evaluated by feedback senders with regard to various factors derived from the media richness theory (Daft and Lengel 1986) and the technology acceptance model (Davis et al. 1989). We explored how the perceived characteristics of feedback features and additional factors like feedback topic or device (e.g., mobile device, PC) affect what feedback feature(s) feedback senders choose to give their feedback to the software company.

Our research results provide new and in-depth insights into the needs and preferences of feedback senders regarding the tool-supported communication between users and software providers and are the basis for the adjustment of embedded feedback tools, ultimately supporting and motivating users to provide feedback for software evolution.

The remainder of the paper is structured as follows. Section 2 gives an overview of the state of the art, including how previous research and practice use different feedback features to gather feedback from users. The section also overviews selected theories that address the capabilities of feedback features and their relations with the use of the feedback channel. Section 3 discusses the applied research methodology, including research questions and research design, and Section 4 illustrates the results to answer the research questions. Section 5 discusses the obtained results, including contribution, implications, and constraints. Section 6 summarizes and concludes the paper.

2. Background

2.1. Feedback Features in Research and Practice

According to the user-to-developer feedback model by (Gallivan and Keil 2003), the feedback communication process between users and the software providers, in particular, the product development team including developers, requirements engineers and other internal stakeholders is characterized by four steps. First, the user (i.e., feedback sender) needs to become aware of a feedback issue she wants to communicate to the product team (i.e., feedback receiver) (Step 1). Second, she communicates (i.e., transmits) her message to the team (Step 2). For that, she needs to be aware of opportunities to communicate her message, select a feedback communication channel and finally transmit her message. The availability of the feedback communication channel but also the channel's richness (see Section 2.2) might affect the choice. Third, the product team receives the information sent by the feedback sender and interprets the message (Step 3). Finally, the product team prioritizes the issues and decides on actions (Step 4).

For Step 2, the user can communicate her feedback by using various feedback communication channels such as phone, ticket systems, personal meetings or dedicated feedback tools (Pagano and Brügge 2013; Stade et al. 2017). Research and practice pay more and more attention to feedback tools that can be implemented as standalone (Heller et al. 2011; Wehrmaker et al. 2012) or embedded feedback tools. For latter, the feedback tool is integrated into the software application and can be activated by the user or the software application itself while the application is running (i.e., while the user is interacting with the software). Some users prefer to initiate the communication of feedback and *push* (Maalej et al. 2009; Schneider 2011) the feedback to the software provider. In this case, the feedback sender triggers the feedback process (*user-triggered*). For that, many feedback tools provide a feedback button visible anywhere in the software, e.g., implemented as a floating button or as a button in the toolbar, that starts the feedback tool. Other users are fine to provide feedback when asked in system-triggered questions (Dennis et al. 2008), so it is the system following rules defined by the software provider who triggers the feedback process and *pulls* (Maalej et al. 2009; Schneider 2011) feedback from its users. This can for example be done with the help of the feedback tool that simply pops-up and asks the users for feedback following predefined rules, e.g., after the website was visited ten times.

Users can communicate feedback as speech act or non-linguistic act (Morales-Ramirez et al. 2015)(see communication act in Table 2-1).

Speech act refers to the communication of feedback in natural language and includes that the feedback sender expresses her feedback in a textual (i.e., written) or an audio (i.e., spoken) communication format (Morales-Ramirez et al. 2015), while non-linguistic acts include visual communication formats such as drawings. For the context of feedback gathering and this paper, we introduce the term *feedback feature*. As Table 2-1 shows, each *feedback feature* is linked to one communication format, but is a more fine-grained description of a communication format and highlights how this format is supported in the feedback tool. One or more feedback features are used by the feedback sender to communicate her feedback issue to the software provider and to create a shared understanding of a situation or task, such as solving a problem for improving the software system. Examples of these feedback features are *text* typed, *screen records* or *emojis*. Feedback tool features support the feedback sender to create and document actual feedback. For example, if the user experienced a usability issue with the website, she can type some free text in a text input field or select a text snippet from a list of problem descriptions, both features provided in the feedback tool. Some feedback tools also ask the user to categorize her feedback and provide a selection feature. Depending on the answer categories, a selection feature can support a textual or visual communication format. If the categories to be selected are represented by words (like *Problem* or *Feature Request*), it is a textual format. A list of icons like a little red bug or a yellow light bulb supports the visual communication format.

Table 2-1 provides an overview of feedback features and categorizes them by communication act and communication format. This table also highlighted the investigated feedback features in this study, including its short names.

Over the past few years, various embedded feedback tools supporting some of the aforementioned feedback features were developed. Table 2-2 shows this variance. It is notable that none of the existing embedded feedback tools supports the screen record feedback feature although requested by research (Hartson and Castillo 1998; Yusop et al. 2015) and promising solutions from other domains already exist, like Loom (https://www.loom.com/) for user research or Lookback (https://lookback.io/) for unmoderated remote usability tests. Moreover, to the best of our knowledge, none of the existing research and commercial tools supports all the feedback features.

Current research about user feedback mainly concerns the feedback receiver side (Step 3 and Step 4 in the user-to-developer feedback model). For example, the usage of machine learning algorithms to automatically classify feedback into categories relevant for software evolution (e.g., (Guzman et al. 2015), visualizations of feedback categorized by topic and sentiment (e.g., tool FAVe (Guzman et al. 2014), approaches for feedback prioritization (e.g., (Kifetew et al. 2017; Seyff et al. 2017)), as well as decision taking and planning (e.g., (Srewuttanapitikul and Muengchaisri 2016). Only a few studies have investigated the feedback sender perspective. While there is indeed research on users' ability to identify and formulate feedback issues (e.g., (Elling et al. 2012; Oriol et al. 2018)) (Step 1), there is a lack of research on *how and why* users choose and use a particular feedback feature (Step 2). Previous research on embedded feedback tools investigated how often feedback tool features were used (e.g., (Elling et al. 2012; Oriol et al. 2018)) or which feedback tool features users expect (Stade and Seyff 2017), but neglected an in-depth analysis. Thus, we still do not know in which situations and for which feedback issues users prefer which feedback feature although such research is requested (Oh et al. 2016).

2.2. Media Richness and Technology Acceptance Model as Underlying Theoretical Frameworks

In this section, we give an overview on two theories namely media richness theory (MRT) and technology acceptance model (TAM) that address the capabilities of feedback features and their relations with the use of the features.

Communication media, such as feedback forms vary in their capabilities to communicate information and develop a shared understanding of a situation (Watson-Manheim and Bélanger 2007). The degree of this capability is known as media richness. A rich media can transmit a sufficient amount of correct information, allow complicated interactions to reduce uncertainty and ambiguity of a situation (Ferry et al. 2001; Sun and Cheng 2007).

Table 2-1. A taxonomy of user feedback.

Communication Act (Morales-Ramirez et al. 2015)	Speech act			Non-Linguistic act			
Communication Format (Morales-Ramirez et al. 2015)	Textual (Written)		Textual (Spoken)	Visual			
Feedback Features*	Free Text Typed[1]	Text Selected[2]	Speech Record[3]	Icons/Symbols Selected (e.g., Emojis, Stars)[4]	Screenshot (Annotated with Drawings)[5]	Screen Record[6]	Further material attached[7]
Investigated Feedback Features in this study**	*Text with Free Text Input (A)*	*Category with Selection of Text Categories (D)*	*Audio Recording with Recording of Voice (F)*	*Rating with Star Rating (B); Emojis with Selection of Emojis (C)*	Screenshot with Taking and Annotating a Screenshot (E)	*Screen Recording (G)*	*Attachment with Upload of an Attachment (H)*

*: Examples of feedback features are: [1]User describes her feedback message in her own words by using a free text input field or a text field overlay / comment bubbles to annotate a screenshot. [2]User selects words/phrases e.g., *Bug* or *Feature requests* to categorize her feedback, selects text modules that describes her issue best, selects a number on a rating scale to judge the overall design of a website. [3]User records her spoken statement. [4]User selects an Emoji to express her current mood or an overall judgment of the application, user choose two out of five stars from a star rating scale to indicate her satisfaction with the application[5]User creates a screenshot and provide annotations like a free hand drawings to highlight areas of the screen [6]User records her screen to show dynamic elements of her screen. [7]User uploads a photo she already took, a receipt, a file created by the application

**: Pranthesis in the last row, refer to Figure 2-1.

Table 2-2. Embedded feedback tools and supported feedback features.

	Free Text Typed	Text Selected	Speech Record	Screenshot (Annotated with Drawings)	Screen Record	Icons/Symbols Selected (e.g., Emojis, Stars)	Attachment	Implementation and Triggers
Research tools								
FAME (Oriol et al. 2018)	Yes	Yes(Categories)	Yes	Yes (Annotations)	No	Yes (Stars)	Yes (Any file)	Mobile and Web app, Push and pull
CAFE(Dzvonyar et al. 2016)	Yes	Yes (Categories)	No	No	No	No	No	Mobile app, Push
QoE probe (Fotrousi and Fricker 2016)	Yes	No	No	No	No	Yes (Emojis)	No	Mobile app, Pull
Infocus (Elling et al. 2012)	Yes	Yes (Categories)	No	Yes(Annotations)	No	No	No	Web app, Push
PaDU (Yetim et al. 2012)	Yes	No	No	Yes(Annotations)	No	Yes (Emoji)	No	Desktop app, Push
No Name (Nakhimovsky et al. 2010)	Yes	No	Yes	Yes	No	No	Yes (Logfile)	Mobile app, Push
OpenProposal (Rashid et al. 2009)	Yes	Yes (Categories)	No	Yes	No	No	No	Desktop app Push
Applications								
Microsoft Office Products (e.g., Word Desktop)	Yes	Yes (Categories)	No	Yes	No	No	No	Desktop app, Push
Google Products (e.g., Keep)	Yes	No	No	Yes (Annotations)	No	No	Yes (Logfile)	Mobile and Web app, Push
Service providers								
Usersnap (usersnap.com) (for customer feedback)	Yes	Yes*(Number rating)	No	Yes (Annotations)	No	Yes*(emojis, stars)	No	Mobile and Web app, Push
Usabilla (usabilla.com (for Apps and Websites)	Yes	Yes*(Number rating,categories)	No	Yes (Annotations)	No	Yes*(Emojis,Star rating)	No	Mobile and Web app, Push
OpinionLab(opinionlab.com) (for Website and App)	Yes	Yes*(Number rating,categories)	No	No	No	No	No	Mobile and Web app, Push

*: refers to the particular configuration of the feedback tool

Media richness theory explains the richness capabilities (Daft and Lengel 1986). Daft and Lengel stated that a feedback feature is considered to be rich when it has the abilities to handle multiple information cues (i.e., The number of ways in which information could be communicated such as written text, voice), allows the sender for tailoring and personalizing the media to her needs, facilitates the provision of immediate feedback, and utilizes verbal and non-verbal language for communication (e.g., body language, facial expression, tones of voice). Studies show that not only the characteristics of a feedback feature, but also other factors may influence the degree that the feature is perceived to be rich. The familiarity of individuals with the media and with each other (Carlson and Zmud 1999), and social influences (Schmitz and Fulk 1991) are some influential factors studied. Moreover, Anandarajan et al. (2010) showed that not only media richness but also the usefulness of the media possibly influence the use of the instant messaging feature.

In the context of communicating user feedback, we argue that a feedback tool is information technology. The technology acceptance model (TAM) is one of the most utilized models in studying system usage (Davis et al. 1989). As indicated by the TAM, the perceived usefulness and ease of use of the feedback tool and its features might influence the intention of users toward using and finally the actual use of the feedback tool. Researchers have already shown that perceived ease of use and usefulness are factors that affect the intention to use a software system (Robert and Dennis 2005).

Although MRT and TAM have been confirmed in several studies, some studies could not fully support these theories (Suh 1999) Dennis and Kinney (1998); (Turner et al. 2010). In this study, we will investigate whether and how the theories discussed are applicable for feedback media in the context of user feedback.

3. Research Design

In this study we are inspired by media richness theory and technology acceptance models to design our study's construct. However, there exist differences in the user feedback context compared to regular communication media such as e-mail. As examples of such differences in the user feedback communication context, the embedded feedback channel in software generally provides one-way feedback communication (i.e., from sender to receiver only). However, media richness theory defines communication as a bi-directional relation. Furthermore, feedback sender and receiver might not have a shared understanding of the receiver's task (e.g., prioritizing feedback, code changes). However, media richness theory assumes that such shared understanding exists, and the individuals are generally aware of the task. Therefore, it is not clear whether the differences may introduce other constraints in the theory, but it would be interesting to discuss to what extent the results of the current study may support the media richness, as a lesson learned from this study.

3.1. Research Objectives

The aim of the research is to understand how feedback senders perceive the characteristics of a feedback feature belonging to a feedback channel embedded in a software application, and how the characteristics impact the use of the feedback channel. By characteristics we refer to *special capabilities or attributes of a feedback feature that distinguish the feature from others*. In this regard, we intend to measure how feedback senders perceived *usefulness, ease of use, explaining a complicated situation, tailoring, transferring emotions* of feedback features, and how they are *familiar* with giving feedback and the feedback feature itself. We also intend to investigate to what extent each factor impacts the intention of the feedback sender to use the feedback channel.

We have investigated the following feedback features: *text, rating, emoji, category, screenshot, audio recording, screen recording,* and *attachment*. These feedback features are provided in industrial and research feedback tools or are requested feedback features, as shown in Section 2.1.

3.2. Research Questions

To achieve the research objectives, the following two research questions are studied:

RQ1: How do feedback senders characterise various feedback features in an embedded feedback channel?

RQ2: What is the relationship between the characteristics of a feedback feature and the use of its feedback channel?

In RQ1, we investigate how feedback senders perceive the characteristics of different feedback features namely *ease of use, usefulness, explaining a complicated situation, tailoring, transferring emotions and familiarity* in an embedded feedback channel. In RQ2 we study whether the characteristics impact the use of a feedback channel. We will measure either a past use of a feedback feature or an intention to its future use.

3.3. Research Method

We conducted the research using a questionnaire-based survey method (Gideon 2012). We designed the research as follows:

3.3.1. Research Conceptual Model

In the first step, we designed a conceptual model to guide data collection and analysis. Figure 2-1 presents the conceptual model that we studied in this research and clarifies how we answer each research question. According to the model, we measured how feedback senders perceive the main characteristics of the feedback features and also investigated how the characteristics are related to the intention to *use* the feedback channel, in general. In the following, we explain how the conceptual model has been formed.

We measured *ease of use* and *usefulness* of the feedback features. Both have shown to be influential characteristics on the use of a feature (Anandarajan et al. 2010). We defined ease of use for the respondents as *the degree to which you believe that using the feedback feature would be free of physical and mental effort*. We defined the usefulness as *the degree to which you believe that using the feature would enhance your performance for providing the feedback as you intend*. We investigated the effect of *ease of use* and *usefulness* characteristics on the intention of users to use the feedback channel.

We also measured the capabilities of the feedback features for *tailoring* and *transferring emotions* as influential factors on media richness. *Tailoring* is defined as *the degree that the user can customise and personalise her feedback using the feedback feature*. We defined transferring emotions as *the degree that the user can express her feelings and the feedback receiver understands her feelings*. Additionally, we measured the capability for *explaining complicated situations* as the effect of a rich media that contributes to the reduction of uncertainty and ambiguity of situations. Since perceived media richness affects the intention to use the media (as discussed in Section 2.2), we expect that the capability for *explaining a complicated situation* can also influence the intention to use the feedback channel. Furthermore, we measured *familiarity*, among other factors discussed in Section 2.2, that may influence media richness (Carlson and Zmud 1999). We also expect that experience affects the intention to use. Therefore, we investigated both relations in the conceptual model presented in Figure 2-1.

We ignored two factors discussed in the media richness theory: multiplicity of cues and immediacy of feedback. Each feedback feature has already known cues, for example, *audio recording* can support voice inflection and *text* can only support written words. Therefore, we did not investigate the multiplicity of cues. We also ignored the immediacy of feedback, as we found it irrelevant to the context of a single feedback feature.

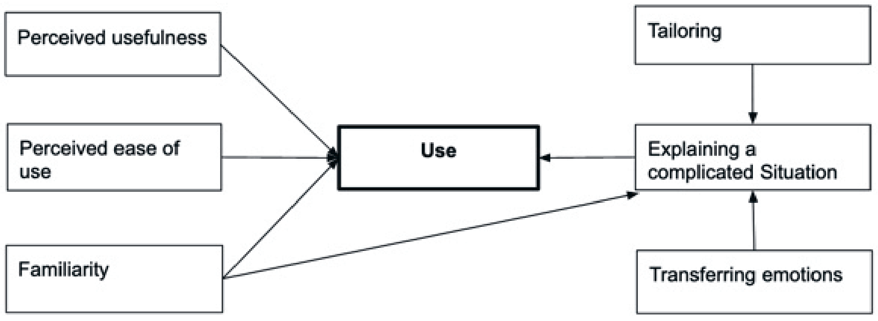


Figure 2-1. The study's conceptual model

3.3.2. Respondents

In order to achieve an adequate sample size to answer our research questions, we decided on 100 participants. We used an international public panel (http://www.consumerfieldwork.com/) to recruit respondents, who were already registered in the panel. It is important to notice that the panel invites paid respondents randomly. The panel claims to have a high response rate and high answering quality due to its active panel management and rigorous quality control.

We decided that as selection criteria, the respondents should have a good level of English knowledge and should be comfortable with reading and writing in English. Furthermore, we defined that one sub-group of respondents (a maximum of 25%) should have seen a feedback form before but should not have provided feedback yet. Likewise, another sub-group (a maximum of 25% of respondents) should not have seen a feedback form yet. This means that at least 50% of respondents should have already provided feedback with a feedback form. The reason for the selection was that we needed to measure the effect of familiarity of the respondents with giving feedback. However, we expected that the users who have already provided feedback could have a better reflection on the rest of the questions. This motivated us to collect more answers from experienced users.

3.3.3. Data Collection Methods

We designed the questionnaire and collected answers using QuestionPro (https://www.questionpro.com/), a survey platform that allows flexible designing of the questionnaire, maintaining data, and generating some reports. For data collection, we created a questionnaire with the aim of identifying the characteristics of feedback features that the respondents perceived (RQ1) and checking whether the characteristics impact their past or intention to use a feedback tool in future (RQ2). The questionnaire is available online at https://bit.ly/2Vyk6Wd.

We started the questionnaire with a short introduction and two questions for qualifying the respondents with English proficiency (see selection criteria above). The questionnaire described the communication of feedback and the overall goal of the study for those participants who qualified with the language criteria. It asked a question whether the respondents have provided feedback in the past (see selection criteria above). Those who indicated that they had provided feedback in the past (i.e., familiarity with giving feedback), were asked to mention the frequency of providing feedback, the relevant feedback context as well as their motivations for giving feedback. Those who had seen a feedback form before but had not provided feedback with such a feedback form were asked to explain why not.

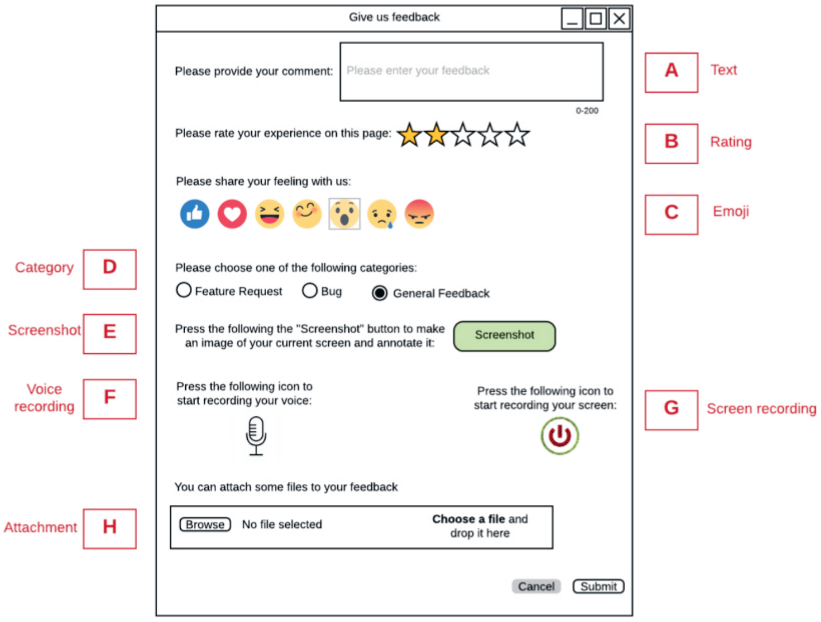
In the next section of our questionnaire, we introduced an exemplary embedded feedback tool. As highlighted in Table 2-2, none of the existing embedded feedback tools supports all the feedback features under investigation. Thus, inspired by existing feedback tools and considering their design (e.g., wording, order of feedback features, layout), we created a mock-up that illustrated all the eight features under investigation. Figure 2-2 presents the sketch that included the following feedback features: free text input (A), star rating (B), selection of emojis (C), selection of text categories (D), taking and annotating a screenshot (E), recording of voice (F), screen recording (G) and upload of an attachment (H). With the help of that screen we wanted to ensure a shared and equal understanding of the feedback features descriptions and the related questions in the online questionnaire.

We designed the next questions according to the conceptual model presented in Figure 2-1. We asked the respondents to state their level of agreements whether the example feedback form (presented in Figure 2-2) in total and for each feedback feature separately is *easy to use*, *useful* and whether they would intend to *use* it. The level of agreement asked in several questions ranged: strongly agree (5), agree (4), neutral (3), disagree (2), strongly disagree (1), no idea (0). We asked the respondents to state the level of agreement about the capabilities of each feedback feature for *explaining complicated situations*, *transferring emotions* and *tailoring*. Meanwhile, in several open-ended questions, we asked participants for the motivation of their choices.

We designed three scenarios that each explained a situation relevant to fixing bugs, requesting a new feature, and feature improvement respectively. The first scenario was about a messaging service where sometimes the characters were wrongly displayed with question marks and sometimes the search functionality did not properly work. The users were supposed to report the problems. The second scenario was about a video streaming service, and the feature about marking watched episodes was missing. The users should report the request for the new feature. The third scenario was about an online shop where the users were supposed to give feedback for improving the user interfaces.

The respondents were asked to choose their preferred feedback features (at least one), given that they intend to give feedback about the situation. The respondents should provide the answers for two cases; either when they are using a smartphone or tablet without a mouse or keyboard or a PC or laptop with a mouse and keyboard. We also asked about the *familiarity* with the context of the scenarios.

In the next to last section of the questionnaire, we asked the respondents to describe in their own words in which situation they would prefer to use each of the feedback features. Also, we requested them to mention any other feedback feature that they would like to use to communicate feedback. Then we asked about their preferences of giving feedback in two triggering situations; when the respondent triggers (i.e., push) or when the system triggers (i.e., pull) the feedback communication process. Finally, a few questions asked demographic information such as education and skills.


Give us feedback

Please provide your comment: Please enter your feedback

0-200

Please rate your experience on this page: ★★☆☆☆

Please share your feeling with us:

Please choose one of the following categories:

○ Feature Request ○ Bug ● General Feedback

Press the following the "Screenshot" button to make an image of your current screen and annotate it:

Press the following icon to start recording your voice:

Press the following icon to start recording your screen:

You can attach some files to your feedback

Browse No file selected Choose a file and drop it here

Figure 2-2. Screenshots of a feedback channel shared in the questionnaire

3.3.4. *Data Analysis Methods*

Research questions were answered using quantitative and qualitative analysis of the answers to the questionnaire. We used descriptive analysis to statistically explain data quantitatively and used PLS-SEM (Partial Least Square-Structural Equation Modelling) analysis (Hair Jr et al. 2016) to statistically find the relationships discussed in the conceptual model (see Figure 2-1). We used an inductive content analysis (Elo and Kyngäs 2008) for analysing and coding the qualitative answers to the questions.

The descriptive analysis and content analysis of respondents' comments regarding the characteristics of feedback features could answer RQ1. The model that we captured using PLS-SEM could answer RQ2.

Descriptive analysis: We performed descriptive analysis statistics, including Min, Max, Mean, and Standard Deviation. We also used *Kruskal-Wallis H* test and independent two samples *t-test* for comparing two means. Using the non-parametric *Kruskal-Wallis H* test, we compared the mean given for the evaluation of feedback features across age groups and educational groups in order to study the effect of age on the choice of feedback features. We chose *Kruskal-Wallis H* test rather than one-way variance analysis (ANOVA) due to lack of homogeneity of the variances. We used the parametric t-test, due to normal distributions of samples, for comparing the sampling distribution of the differences between system-triggered and user-triggered approach. *Kruskal-Wallis H and t-test* were done at the significance level p<.05. We used the tool IBM SPSS Statistics version 26 for performing the analysis.

PLS-SEM analysis: We created a PLS-SEM path model using SmartPLS software version 3.2.9. PLS-SEM is one of the second-generation statistical methods since the early 1990s that is commonly used in social science research (Hair et al. 2019). This is a multivariate analysis to model and relate the set of multiple dependent and independent variables. The literature recommends PLS-SEM modelling is well suited for validating exploratory models (Hair et al. 2019). Therefore, the analysis technique fits well to model the constructs that we discussed in Figure 2-1 and consequently answer RQ2. The PLS algorithm does a sequence of regressions in terms of weight vectors. The weight vectors are obtained iteratively to convergence or reaching the stop criterion. We prepared the first model in the Smart-PLS tool and ran the algorithm using the default setting of the tool (300 maximum iterations and the stop criteria of 7). The model shows the strength of the relations between indicators and their latent variable, called outer loadings. An outer loading represents the absolute contribution of the indicator to the definition of its latent variable. The model also shows the constructs and the strengths of their relations, with a coefficient value. We evaluated the generated path model following Hair et al. (2019)'s guideline and tuned the model correspondingly. We evaluated two steps; evaluation of the measurement model, and evaluation of the path structural model.

Evaluation of the measurement model: As the first activity of evaluating the measurement model, we removed the indicators with outer loadings below 0.4 and kept the indicators with the loadings above 0.7, as suggested by (Hair Jr et al. 2016). For the loadings between 0.4 and 0.7, we conceptually evaluated the importance of each indicator for the construct. For example, we kept the indicator relevant to the usefulness of *text* in the model although its loading was 0.516. The reason was that literature could confirm the *usefulness* of *text* and many respondents qualitatively mentioned its usefulness in their answers to the questionnaire. We were inclusive unless we needed to improve *average variance extracted (AVE*). We evaluated AVE for each construct in the second step of the assessment of the measurement model. The step addresses the convergent validity to explain the extent to which the construct converges to explain the variance of its indicators. An acceptable AVE value is 0.50 or higher. The third step was assessing the internal consistency reliability, using the c*omposite reliability* measure. The values between 0.70 and 0.95 range from *satisfactory to good*. The fourth step is to evaluate discriminant validity. It is used to identify the extent to which the construct is distinct from other constructs. For this, we measured *heterotrait-monotrait (HTMT).* The measure has a threshold value of 0.90.

Evaluation of the structural model: As the first activity, we assessed collinearity of the constructed model using *VIF (Variance Inflation Factor)* measure to understand whether redundant indicators are used to measure two (or more) constructs, and therefore eliminate the redundant indicators. *VIF* of 5 and higher indicates a potential collinearity problem. Then we evaluated the explanatory power of the model using the most commonly used measure called R-square. R-square measures the variance of an endogenous construct (i.e., the construct that is explained in the model) explained by all the linked exogenous constructs (i.e., constructs that explain other constructs) to identify if different exogenous constructs predict the same endogenous construct. The R-square value ranges from 0 to 1, with higher values indicating higher levels of accuracy. Finally, we measured the *indirect effect* to understand the relevance of other constructors, particularly in explaining the *Use* construct. Indirect effects show the relation that has at least one intervening construct.

Inductive content analysis: We performed initial coding of the answers to the open-ended questions by underlining all relevant terms to the capabilities of features and motivations for the use of the features. We grouped the initial codes to form final codes considering meaning relevancies, being synonyms or having similar stems. Then we created categories and assigned each quote to a category. We, the first and second authors, developed the categories independently, and decided the final categories in a joint meeting. In the last step, we summarize the capabilities of the feedback features based on the provided categories.

4. Analysis and Results

4.1. Demographic Information

A number of 231 respondents started the questionnaire and 100 forms were completed. The rest did not pass the attention checking questions or stopped intentionally. The respondents completed the questionnaire in an average of 23 minutes ([18.21, 27.6] minutes, 95% confidence interval).

Among the respondents, 77% evaluated their English proficiency as very good, 20% as good and 3% as acceptable. The respondents were aged between 18 to 74 years old; 18-34 (20%), 35-54 (52%), and 55-74 (28%) years old. The distribution of respondents' educational degrees was as follows: No university degree (26%), Associate/Bachelor degree (47%), Master/professional degree (23%), PhD (4%). 82% of the respondents did not have knowledge or experience related to designing, coding, testing, selling, supporting, managing software applications or conducting research within the given topics. Among the respondents, 77% had already experienced providing feedback in an embedded channel, 12% had seen such feedback forms but have not provided any feedback and 11% had never seen such feedback forms.

4.2. The Characteristics of Feedback Features in an Embedded Feedback Channel (Answer to RQ1)

4.2.1. Quantitative Analysis

Figure 2-3 shows the capabilities of the feedback features perceived by the respondents. The figure presents the extent to which the respondents perceive the capability of each feedback feature (i.e., *text*, *rating*, *emoji*, *category*, *screenshot*, *audio recording*, *screen recording*, and *attachment*) for explaining complicated situations, tailoring the feedback as they intend, and transferring emotions. The respondents evaluated the capabilities in the range of strongly agree (5) to strongly disagree (1); 0 refers to having no idea. The detailed descriptive analysis of their evaluation (Min, Max, Mean, Standard deviation) has been presented in Table 2-7 in Appendix.

Feedback Characteristics Analysis: Respondents evaluated each feedback characteristics as follows:

Usefulness. The results indicate that the respondents perceived *text* (93%) and *rating* (84%) as useful features.

Ease of use. They also agreed (i.e., rated strongly agree and agree) that *text* (93%) and *rating* (82%) are easy to use. *Emoji* (80%) was also recognized as easy feedback. *Familiarity.* Most of the respondents were *familiar* with *text* (93%) and *rating (95%)*, while most of them were unfamiliar with *audio recording (62%)* and *screen recording (52%)*.

Explaining a complicated situation. Most of the respondents disagree (rated disagree and strongly disagree) that *rating* (52%), *emoji* (69%) and *category* (42%) are good feedback features for *explaining a complicated situation*.

Tailoring. The results indicate that *text* is perceived to be the best method for *tailoring* the feedback. 93% of respondents strongly agreed or agreed for the *tailoring* capability of *text*. *Emoji* is not a proper feature for *tailoring*, among others as only 35% of respondents rated it strongly agree and agree.

Transferring emotions. Apart from *text* that was found good for *transferring emotions* (89%), *emoji* (81%), *rating* (60%) and *audio* (59%) were also perceived good. However, the respondents did not find that *screen recording,* and *screenshot* are suitable for *transferring emotions*, as only 25% and 26% of respondents rated them very good and good respectively.

The results illustrated in Figure 2-3 show that overall *text* is perceived to be the most capable feedback feature among the others. Text feedback is a good approach for *explaining a complicated situation*, *transferring emotions* and *tailoring* the feedback. 93% of respondents agreed that *text* is good for *tailoring* feedback. The results surprisingly showed 94% and 89% of the respondents found *text* suitable respectively for *explaining a complicated situation* and transferring emotions. 93% of respondents were familiar with *text* and found it an easy and useful feedback feature.

Scenario-wise Analysis: As discussed earlier, we defined three scenarios. For each scenario, we asked the respondents what feedback features they would use considering the device: i) a smartphone or tablet, ii) PC or laptop. We presented the results in Table 2-6 in Appendix. The results show that for all three scenarios, *text* feedback was the most preferred feedback feature (98%, 95%, and 90% in scenarios 1-3 respectively). *Audio recording* and *screen recording* were the least preferred feedback feature (43% and 45% for scenario 1). The results indicate that when the respondents would use a PC or laptop, they preferred giving *text, screenshot, screen recording,* and *attachment* feedback. However, on a smartphone or tablet, *emoji* and *audio recording* are the preferred features.

Age-wise Analysis: The results suggest that there are no significant differences across categories of age for most of the feedback features. We categorized the respondents into three age groups: (1) 18-34, (2) 35-54, and (3) 55-74 years old. As an example, the analysis for *usefulness* was as follows: (Kruskal-Wallis: H(text)=2.09, p=0.35; H(rating)=0.049, p=0.98; H(emoji)=0.504, p=0.78;

H(screenshot)=0.420, p=0.811; H(audio recording)=1.045, p=0.59; H(screen recording)=0.795, p=0.67; H(attachment)=7.535, p=0.02). *Attachment* was the only feedback feature that the results suggest a significant difference in *usefulness*, *explaining a complicated situation* and *transferring emotions*. For example, the Mean (M) that respondents assigned for *usefulness* of *attachments* was differently valued (age group 1: M=3.5; age group 2: M=3.48; age group 3: M=4.11). We could not find a plausible way to interpret the differences.

Education-wise Analysis: The results suggest that there are no significant differences across categories of education for the feedback features. We categorized the respondents into five educational groups: (1) Less than a high school diploma, (2) high school degree or equivalent, (3) Associate degree/Bachelor degree (4) Master degree/Professional degree and (5) Doctorate. As an example, the analysis for *usefulness* was as follows: (Kruskal-Wallis: H(text)=2.396, p=0.66; H(rating)=6.707, p=0.15; H(emoji)=7.849, p=0.10;

H(screenshot)=5.016, p=0.29; H(audio recording)=3.908, p=0.42; H(screen recording)=4.405, p=0.35; H(attachment)=3.901, p=0.42).

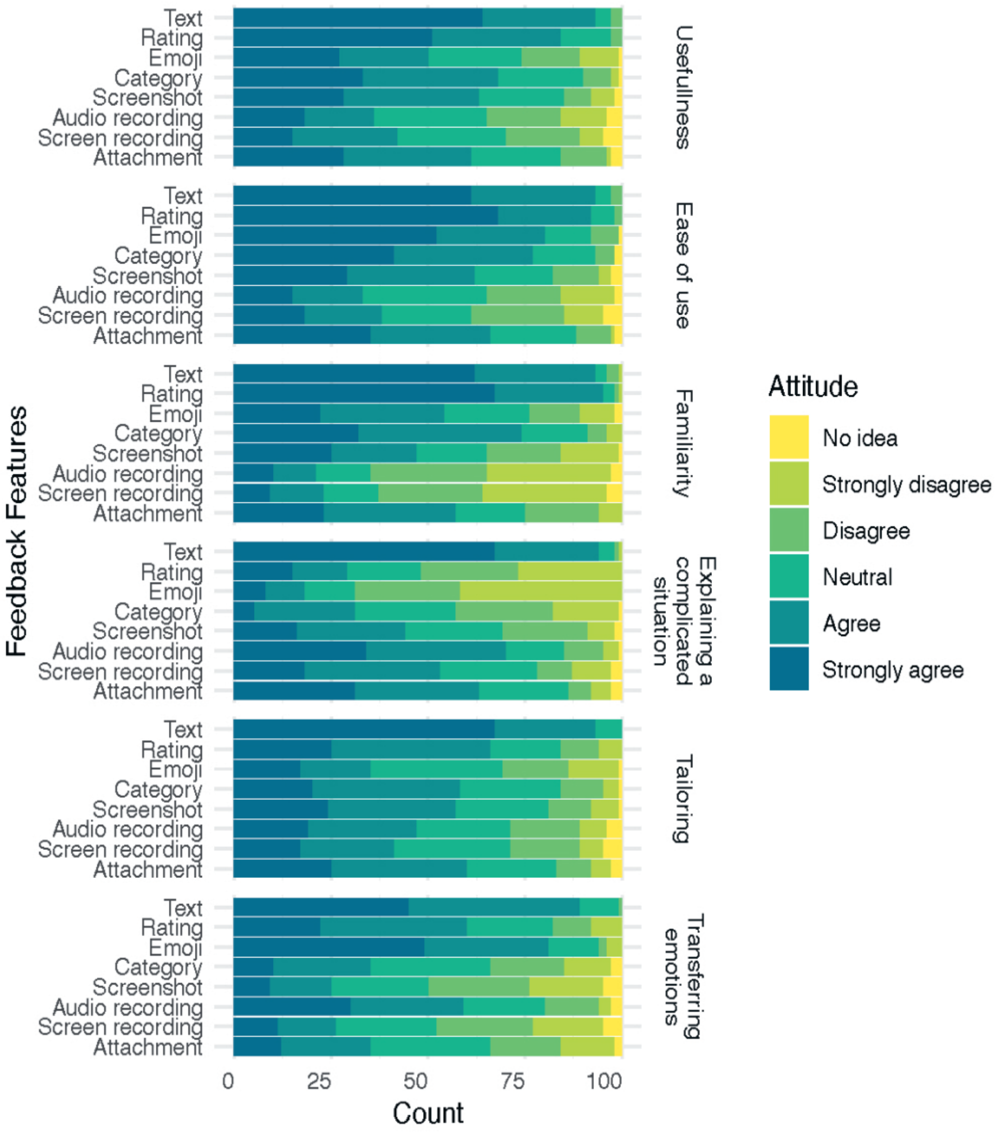


Figure 2-3. Characteristics of Feedback Features (Respondents' rankings)

4.2.2. Qualitative Analysis

The qualitative analysis of the respondents' free text comments indicates how the respondents characterise the feedback features. During the analysis we categorised the answers of the respondents to the open-ended questions, and summarised them as follows, while providing examples of respondents' quotes.

The answers show that respondents usually use *text* when they intend to explain a situation in detail. Many of the respondents stated that they use *text* in most situations because it is easy and flexible and can support giving fast feedback. One respondent reasoned using text because this feature has flexibility to report short statements, using just a few words.

> *"This is my preferred form of feedback for most scenarios where I can easily explain the issue and also use my words to demonstrate how I feel (my emotions) if that is relevant to the feedback I am providing. I just feel that free text allows you to express yourself and take the time to do so, as I am doing now!"*

Several answers mentioned that the respondents use *text* feedback if they have a keyboard while giving feedback.

> *"I will use text feedback when I have the possibility to use a keyboard."*

The answers show that *rating* is used to share feelings and suitable for giving a generic evaluation of the application when there is no need for sharing detailed information. Rating is perceived to be easy and a quick way of communication. Some respondents mentioned that they use *ratings* only if they are extremely satisfied or dissatisfied. Some respondents mentioned that they do not use rating because it may convey vague information. One respondent would use *rating* only together with *text* feedback. *Rating* feature is preferred on smartphones (when a physical keyboard is not available to write text).

> *"Where I am being asked to rate the suitability/efficiency of existing applications - an instinctive approach rather than measured."*

> *"Only a rating without text is just enough when everything is perfect, and I want to give let's say '5 stars' to just show how happy I am with an app for example."*

The answers show that *emoji* is evaluated similar to *rating*; suitable for sharing feelings and emotions, a quick and easy way of giving feedback. However, many respondents believe *emojis* is not professional and conveys more fun characteristics. They believe that *emoji feedback is* subjective, the mood of respondents may affect the emoji choices and could be difficult for feedback receivers to interpret them.

> *"Don't really use the feature as it is a bit unprofessional when contacting a company, [we] use only with friends."*

The respondents provide *category* feedback to respond to the company's inquiries regarding the classification of feedback that the feedback receivers found useful for the product development. The respondents found the *category* feedback easy and quick to use. On smartphones, the respondents chose to use the *category* feedback.

> *"It is a very good possibility to narrow down the issues for the correct support department."*

The answers show that *screenshot* is used to provide evidence for better and clear explanation and illustration of technical issues visible on a screen such as a notification or error message, a user interface issue or an unknown behavior of the software. *Screenshot* was recognised as a better way of giving feedback than *text*, when the problem is complicated to explain while making a *screenshot* makes it understandable for the feedback receiver like the helpdesk. One respondent referred to the privacy issue to motivate why she does not use the feature.

> *"I have only ever used this to show a technical problem with a website or to prove an action I carried out on a website which went wrong/or which is being disputed by the website."*

4.3. Factors Affecting the Use of a Feedback Channel (Answer to RQ2)

We modeled the relationship between the feedback characteristics and the use of the feedback channel using the PLS-SEM analysis. The model is based on the conceptual model presented in Figure 2-1 where we measured the characteristics of feedback features including *usefulness* (i.e., perceived usefulness), *ease of use* (i.e., perceived ease of use), *familiarity* (i.e., previous experience), *explaining a complicated situation*, *tailoring* and *transferring emotions* using the choice questions discussed in Section 3.3.3. We also measured *usefulness*, *ease of use* and *familiarity* of the feedback channel presented in the questionnaire.

Figure 2-4 shows the structural equation model, where the circles present a construct and rectangles show the indicators of the construct (i.e., the variables that make the construct). The name of the constructs starts with a capital letter. The model conveys two sub-models; A *structural model* in the context of PLS-SEM represents the constructs and the relationships between the constructs; and a *measurement model* of the constructs displays the relationships between the constructs and the corresponding indicators. We defined the *measurement model* reflective (the arrows are from constructs to indicators) as we think that logically we also could replace the indicators in the model (e.g., to consider other feedback features).

The number presented on an arrow between two constructs shows the coefficient level of the relation indicating the strength of the relationship between the two constructs. The coefficient level is between 0 to 1. The closer this value is to 1 the stronger the relationship is. This also means that the value close to 0 indicates no relationship. The number on an arrow between an indicator and a construct identifies the loading of the relation (i.e. called outer loading) to identify the indicators' contribution to the construct or in other words this value indicates how the construct is reflecting on its indicators. Similarly, to the coefficient level, valid values for loading are also in the range of 01. Values below 0.3 indicate that the indicator has no absolute contribution to building its assigned construct, which normally is not considered in the model. Values between 0.4 and 0.7 still indicate a weaker contribution and here literature suggests discussing whether to keep the indicators or not.

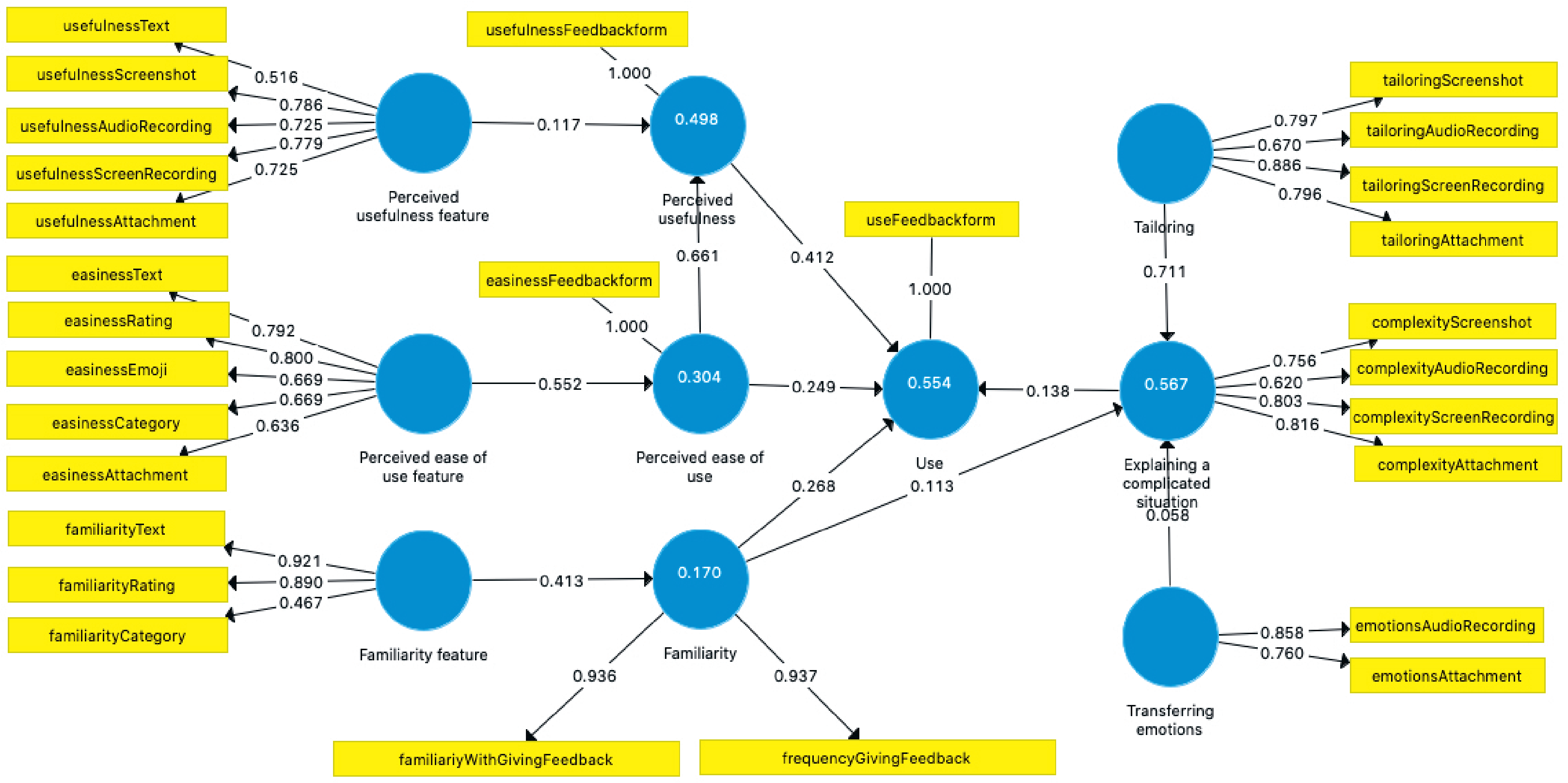


Figure 2-4. PLS-SEM path model of influential factors on Use

Table 2-3. Evaluation of PLS-SEM path model (NA: Not Applicable)

Constructs	**Indicators variables**	**Measurement Model Assessment**				**Structural Model Assessment**		
		Convergent validity		**Internal consistency reliability**	**Discriminant validity**			
		1-Loading (>0.708)	2-AVE (> 0.50)	3-Composite reliability (> 0.70)	4-HTMT (<0.90)	1-VIF (<3)	2-R-squae (0< , <1)	3- Total indirect effect (on Use)
Use (intention to use)	useFeedbackform	1.000	NA	NA	Yes	1.000	0.554	NA
Perceived usefulness	usefulenssFeedbackform	1.000	NA	NA	Yes	1.000	0.498	0.000
Perceived usefulness feature	usefulnessText usefulnessScreenshot usefulnessAudioRecording usefulnessScreenRecording usefulnessAttachment	0.516 0.786 0.725 0.779 0.725	0.509	0.835	Yes	1.0877 1.9059 2.9392 2.2241 1.5609	NA	0.0478
Perceived ease of use	easinessFeedbackform	1.000	NA	NA	Yes	1.000	0.304	0.272
Perceived ease of use feature	easinessText easinessRating easinessEmoji easinessCategory easinessAttachment	0.792 0.800 0.669 0669 0.636	0.513	0.839	Yes	1.8358 1.5025 1.6351 1.4349	NA	0.288
Familiarity	familiarityWithGivingFeedback frequencyGivingFeedback	0.936 0.937	0.877	0.935	Yes	2.3185 2.3185	0.171	0.016

Constructs	Indicators variables	Measurement Model Assessment				Structural Model Assessment		
		Convergent validity		Internal consistency reliability	Discriminant validity			
		1-Loading (>0.708)	2-AVE (> 0.50)	3-Composite reliability (> 0.70)	4-HTMT (<0.90)	1-VIF (<3)	2-R-squae (0< , <1)	3- Total indirect effect (on Use)
Familiarity feature	familiarityText familiarityRating familiarityCategory	0.921 0.890 0.467	0.620	0.820	Yes	1.9538 1.7507 1.2139	NA	0.117
Explaining a complicated situation	complexityScreenshot complexityAudioRecording complexityScreenRecording complexityAttachment	0.756 0.620 0.803 0.816	0.567	0.838	Yes	1.6100 1.2030 1.6564 1.6153	0.567	0.000
Tailoring	tailoringScreenshot tailoringAudioRecording tailoringScreeRecording tailoringAttachment	0.797 0.670 0.886 0.796	0.625	0.869	Yes	1.7629 2.3045 1.3662 1.6525	NA	0.098
Transferring emotions	emotionsAudioRecording emotionsAttachment	0.858 0.760	0.657	0.792	Yes	1.1127 1.1127	NA	0.008

All values higher than 0.7 indicate a strong contribution of an indicator to building its assigned construct and therefore keeping the indicator in the model.

According to the model presented in Figure 2-4 the constructs *Usefulness*, *Ease of use,* and *Familiarity* show acceptable value for the relationships with *Use* construct. While the constructs *Explaining a complicated situation* and *Transferring emotion* have weak relationships with *Use.* The construct *Tailoring* could not establish a relationship with *Use* while showing a strong relationship with the construct *Explaining a complicated situation.* The relations between indicators and constructs reveal that particular feedback features are stronger in some characteristics than the others. As discussed above, the relations are called outer loading and simply referred to as loading in this report. The results show that *text, screenshot*, *audio recording*, *screen recording,* and *attachment* all contribute to the construct *Perceived usefulness feature,* and we interpret that respondents have perceived these features more useful rather than the others. *Also, text, rating, emoji, category* and *attachment* contributes to the *Perceived ease of us*e, meaning that these features are perceived to be easy to use. The feedback features *attachment, audio recording, video recording* and *screenshot* contributes to constructs *Tailoring feedback* and *Explaining a complicated situation* referring that the features are more capable in *explaining a complex situation* and *tailoring*, while *audio recording* and *attachment* could relate more to *transferring emotion* capability. *Text, rating, category* were recognised as the feedback feature, which the users are more *familiar* with.

The evaluation results of our PLS-SEM model are satisfying. Table 2-3 shows the evaluation results. The evaluation for *composite reliability* (between 0.79 and 0.93) showed that the measurement has a good reliability level. We could not detect any *collinearity* problem, as the *VIF* values all were lower than 0.3. So, we could not detect redundant indicators in the constructs. *R-square* of *Use* (0.554), *Perceived usefulness* (0.498), *Explaining a complicated situation* (0.553), can be considered as high values, also considering that (user) behaviour measurements cannot be accurate (Hair Jr et al. 2016). The results show the *R-square* values of *perceived ease of use* (0.304) and *familiarity* (0.170) are lower than the other *R-square* values. This means that these factors explain less variance in *Use* than other factors investigated. But indirect effect analysis confirms that most characteristics of features including its *perceived ease of use* and *familiarity* indirectly effect on *Use*. In this analysis we could also find a relation (0.364) for perceived ease of user to perceived usefulness, meaning that the easiness of a feedback feature affects its perceived usefulness.

4.4. User-triggered vs System-triggered Feedback

We asked respondents to choose the likelihood that they provide feedback if they have feedback to share in the following two approaches:

i) User-triggered approach (i.e., push); the feedback senders press for example a button labelled *feedback* and ii) System-triggered approach (i.e., pull); the feedback senders give feedback when the system pops up a feedback form automatically to ask for feedback.

The level of likelihood ranged: almost always (5), usually (4), occasionally not (3), usually not (2), almost never (1).

The results as presented in Figure 2-5 show that the likelihood users trigger the feedback form to give feedback is slightly higher (48%) in comparison to when the system triggers feedback form and the users provide feedback (38%). However, more users *occasionally* give feedback in system-triggered approach (43%) in comparison to a user-triggered approach (30%). Also, the likelihood that the users *usually do not* give feedback in a system-triggered approach is slightly lower (19%) than the user triggered approach (22%).

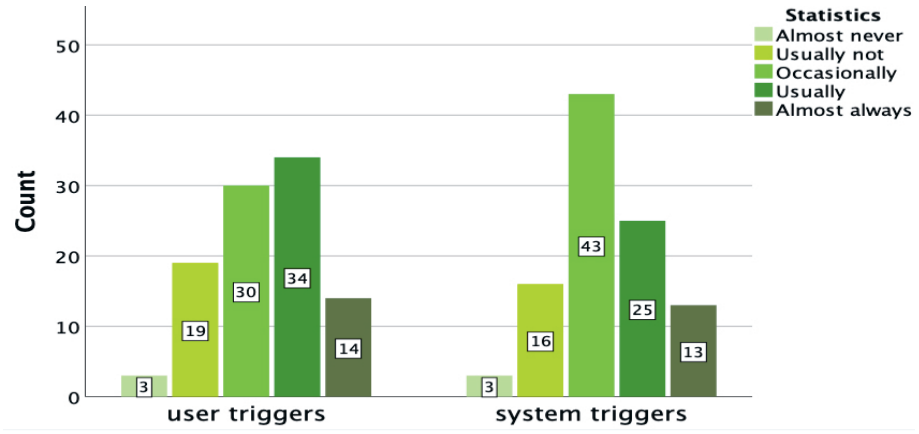


Figure 2-5. User-triggered vs system-triggered feedback

The outcome of the independent *t*-test shows that on average, respondents indicated a greater likelihood to provide feedback given a user triggered approach (*M*=3.37, *SD*=1.04) than given a system triggered approach (M=3.29, SD=0.99). This difference was not significant (*t*(99)=0.56, *p*=.58).

We found that (i) one third of respondents (38%) indicated the same likelihood that they provide feedback given a user or system triggered approach, (ii) one third of respondents (30%) indicated a higher likelihood for a system triggered approach compared to user triggering (mean average of deviation: +1.57), and (iii) one third of respondents (32%) indicated a lower likelihood for system triggering compared to user triggering (mean average of deviation: -1.72).

Figure 2-6 shows the crossed frequencies of likelihoods for both triggering approaches. For ten respondents who would *almost never* or *usually not* provide feedback in a user triggered approach a system-triggering approach would increase their likelihood to provide feedback up to *usually* or *almost ever*. For 12 respondents who would *usually* or *almost always* provide their feedback in a user-triggered approach, their likelihood would decrease to *almost never* or *usually not*.

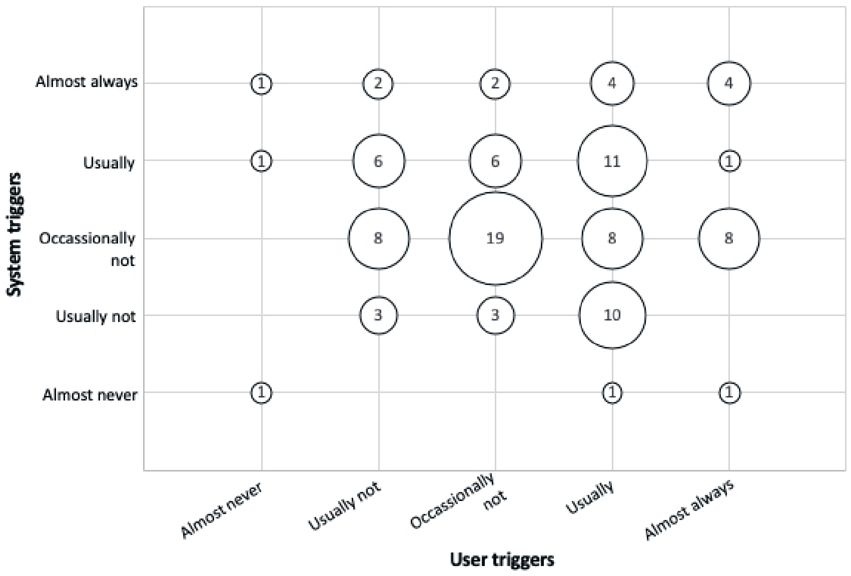


Figure 2-6. Comparison likelihoods for user-triggered vs. system-triggered feedback

5. Discussion

5.1. Implications

The path model analysis (PLS-SEM) of feedback characteristics, summarised in Table 2-4, revealed that *perceived usefulness* of a feedback channel has the strongest relationship with an intention to *use* the feedback channel. When a feedback channel is perceived to be *easy to use*, on one side it will have a positive influence on *perceived usefulness* and on the other side on *use*. Therefore, *ease of use* directly and indirectly would improve the intentions to *use* the feedback channel, which hopefully leads to its actual *use*. *Familiarity* with the feedback features reflects on *familiarity with giving feedback* and *frequency of giving feedback*. Our result could support the effect of familiarity on the use of a feedback channel that (Schmitz and Fulk 1991) has already shown. The findings imply that companies need to consider courses of action (e.g., tutorials) to make their users familiar with the feedback features, engage them to give feedback that could start with simple feedback that only uses one or two feedback features. The *familiarity* will probably impact the *perceived* *ease of use* and consequently improve *perceived usefulness*, ultimately leading to more use of the feedback channel.

Reflecting on our conceptual model the findings confirm that the technology acceptance model (Davis et al. 1989) is applicable to the context of user feedback, however, confirmation of media richness theory needs further investigation in the context of user feedback. Among the factors chosen to measure media richness, *transferring emotions* and *familiarity* have rather weak relationships with *explaining a complicated situation*, which is also weakly related to use of the feedback channel. It implies that either the feedback context does not have conformity with the principles of media richness theory, or we are missing some other factors for measuring e*xplaining complicated situations*. Or a rich feedback feature may show itself in other factors than *explaining a complicated situation*.

Table 2-4. Summary of results PLS-SEM

Constructs	**Coefficient**	**Support**	**Theory relevance**
Perceived usefulness → Use	0.412	Supported	technology acceptance model (TAM)
Perceived ease of use →Use	0.249	Supported	
Explaining a complicated situation → Use	0.138	Supported Weakly	media richness theory (MRT)
Tailoring → Explaining a complicated situation	0.711	Supported	
Expressing emotions → Explaining a complicated situation	0.058	Supported Weakly	
Familiarity → Explaining a complicated situation	0.113	Supported Weakly	
Familiarity → Use	0.268	Supported	

Table 2-5 summarises the strengths and limitations of feedback features for each feedback characteristic discussed in this paper.

Loading represents the absolute contribution of feedback features to the definition of the feedback characteristics and *Mean* shows the average ranking of users for the characteristic. The green boxes show the strength and the red boxes show the limitation of each feedback feature. The grey boxes represent the inconsistencies between the user ranking of a particular feedback feature for a particular characteristic and its contribution to using a feedback tool. We cannot have a proper judgement on this group of data, to categorise as strengths or limitations.

We believe the a success for use of feedback features is to match their capabilities to the users' needs for sharing their feedback. Therefore, a feedback channel that includes several feedback features could be a solution, rather than including only one feature or a small set of features for users to give their feedback. For example, except *attachment*, those feedback features which are perceived to be *easy to use*, are not good in *explaining a complicated situation*. This gives the opportunity for users to switch between the feedback features depending on the issue that they need to communicate. However, what we still do not know is about how users choose when it comes to trade-off between the characteristics. In the above example, we do not know if the users intend to *explain a complicated situation*, whether they choose the feedback features that have the better capability for their explanation or the ones that are *easier to use*.

The analysis of triggering approaches showed that the likelihood that the users give no feedback in a system-triggered approach is quite low but the same as in a user-triggered approach, given that the users are already aware of a feedback issue to communicate. This implies that companies can indeed request user feedback from their users as the chance that users will provide their feedback is quite high. However, the companies may not need to request those users who have the habit of giving feedback, otherwise the users likely reject their requests. We believe the requesting feedback from users with a simple feedback form is a useful approach for motivating more users to provide feedback. However, the request should not disturb users, although the disturbance has a negligible impact on users' quality of experience (Fotrousi et al. 2018).

Table 2-5. Strength and limitation of feedback features*

	Usefulness		**Easy to use**		**Familiarity**		**Explaining complicated situation**		**Tailoring**		**Transferring emotions**	
	Loading	Mean	Loading	Mean	Loading	Mean	Loading	Mean	Loading	Mean	Loading	Mean
Text	0.516	4.58	0.792	4.51	0.921	4.50	-	4.58	-	4.60	-	4.33
Rating	-	2.65	0.800	4.58	0.890	4.59	-	2.65	-	3.69	-	3.56
Emoji	-	2.15	0.669	4.22	-	3.39	-	2.15	-	3.06	-	4.20
Category	-	2.74	0.669	4.07	0.467	3.93	-	2.74	-	3.06	-	2.93
Screenshot	0.786	3.18	-	3.64	-	3.20	0.761	3.18	0.791	3.53	-	2.55
Audio Recording	0.725	3.83	-	2.95	-	2.28	0.611	3.83	0.669	3.22	0.797	3.60
Screen Recording	0.779	3.33	-	2.97	-	2.29	0.804	3.33	0.884	3.13	-	2.61
Attachment	0.725	3.69	0.636	3.84	-	3.49	0.819	3.69	0.804	3.57	0.826	2.95

*: green: strengths, yellow: weaknesses, grey: inconclusive

In general, the study contributes to guiding practitioners and researchers to design an embedded feedback channel while improving its use. We suggest companies consider triggering approaches for collecting user feedback; A feedback form that has multiple feedback features, where the users trigger the form by pushing the feedback button, and a simple feedback form adjusted to the context of the recent use. Having the two approaches enable the users who have the habit of giving feedback to give the feedback when intended, and motivates the users, who do not have the habit, to communicate their feedback just after experiencing it.

5.2. Validity and Reliability

In this section, we evaluated the study to ensure that the survey instrument is measuring what we intended (validity), and the conclusions are reliable (reliability). We also added External validity as we think generalisation is an important aspect to discuss. We categorised the evaluation as (Kitchenham and Pfleeger 2008) defined.

Content validity: The validity concerns appropriateness of the instruments. To check whether the questions are understandable, the third author who was not involved in designing the instrument reviewed the questionnaire. Then we organised a pilot study where seven respondents answered the questionnaire. The respondents were non-professionals and professionals in software engineering research and practice. We received feedback about understandability of the questions in a meeting, or via written and audio messages. We analysed the collected data to assure that the questions are answerable. We updated the questionnaire accordingly.

We used a panel to recruit the respondents. Although a survey with public panels could introduce some threats (Oppenheimer et al. 2009), we considered them while designing the questionnaire instrument. We introduced control questions for checking the English skills of the respondents and three trap questions. The trap questions are not easily detectable among other questions and asked respondents a particular choice (i.e., Please choose Neutral). When the respondents chose the wrong option, the survey was terminated. Two trap questions were in the middle of the questionnaire and one at the end. The trap questions allowed us to exclude the inattentive respondents who did not focus on giving their answers. Furthermore, the panel claimed to have a rigorous quality control procedure. The respondents of the panel receive only a few invitations per month and also much better incentive per survey. As the panel promised, the panel filtered bad respondents after each survey they participated. The respondents mostly have learned to become good respondents and answer thoroughly as the panel silently blocks the respondents over time if they are not good.

Criterion validity: The validity concerns the ability of the instrument to distinguish respondents for different groups. The respondents belong to groups of different ages, education, and proficiency. We did not provide a filter while inviting the participants. Instead, we assured that the invitations introduced randomness to allow us to use analysis methods that require randomness. However, we classified the answers during the analysis.

Construct validity: The validity concerns the extent to which the instrument measures the construct it is designed to measure. We prepared a conceptual model based on TAM and MRT theories and used the model to test data in PLS-SEM analysis. We collected sufficient samples (i.e., 100 answers to the questionnaire) for the PLS-SEM analysis. The sample size should be at least ten times the largest number of arrowheads (indicators) at anywhere in the PLS path model (Barclay, Higgins, & Thompson, 1995). As we presented in Section 4.3, the largest indicator group used to measure a single construct is 4 now. Therefore 100 samples collected in the study are appropriate.

External validity: This concerns the extent to which the results of a study can be generalised for other studies. The population of the study was public mainly from the UK, Austria, and Germany. We categorised the responses based on different age and education. Despite the lack of homogeneity for some groups of samples (e.g., 52% of our sample aged from 35 to 54), our analysis could not detect any significant differences between groups. Therefore, we can generalise our results within the samples collected from these countries. People from other countries might have different cultures or attitudes toward using a feedback form. For example, emoji was not well received by the respondents and one even called the Emojis *childish*. Data from all around the word is needed to generalise this

Inter-observer reliability: A threat to reliability is a biased observer during analysis. For the qualitative analysis, the first and second authors independently coded open-ended questions and interpreted the results. Then in a joint meeting discussed the conflicts and concluded the findings.

5.3. Limitations and Future work

In this study, we focused on known feedback features of embedded feedback tools; however, there are also other types of features, including video conferencing and screen sharing. Research can replicate this study and extend it for other feedback features and populations from other countries, particularly Non-EU citizens. Furthermore, this study focused on the characteristics of feedback features, but we did not study how the combination of different feedback features can be perceived, which can be studied in future.

This study addresses the characteristics of feedback features from the feedback sender perspective. It is worthwhile to explore how the characteristics that impact the quality of information communicated are perceived and evaluated by feedback receivers. We collected a few data about combined feedback features, but our overall approach was to investigate the feedback features separately. This study can be extended in future to investigate how the combination of user feedback features influence the use of feedback features and consequently, the use of feedback tools.

The study was limited to just three scenarios evaluated in the survey. Future studies can extend the scenarios and consider other contexts. Environmental factors like walking, having people around might also be candidates for future investigations.

6. Conclusion

In the context of software evolution, the provision of a suitable feedback channel for users is crucial to allow them to communicate their feedback as they need. However, little is known about the characteristics of the feedback features in an embedded feedback channel and how the characteristics relate to using the features. In this study, we aimed to understand i) the characteristics of feedback features in an embedded feedback tool, ii) whether the characteristics impact the use of feedback features, and iii) strengths and limitations of feedback features. To achieve the aims, we surveyed 100 public respondents. We chose six factors for evaluating feedback features by inspiration from technology acceptance model and media richness theory, namely *usefulness, ease of use, familiarity* level of users with the feedback feature and giving feedback in addition to capabilities for *explaining a complicated situation*, *tailoring feedback, and transferring emotions.* In total, we selected eight feedback features including *text, rating, emoji, category, screenshot, audio recording, screen recording,* and *attachment*, in which the respondents of the survey in the role of feedback senders evaluated them. The results confirmed that *perceived usefulness, ease of use*, and *familiarity* (e.g., with giving feedback, and with feedback features) are essential factors affecting the *use* of a feedback channel. Although the study could confirm *text* is the most preferred feedback feature, there are other feedback features such as *screenshot*, and *audio recording* that have more influential contributions on *explaining a complicated situation* and *tailoring* the feedback to the needs. Overall, the study suggests providing feedback forms with multiple feedback features to allow users when they push the feedback button to be able to choose among the features based on their preferences and needs. As a complementary approach, we also suggest companies design simple feedback forms (with limited feedback features such as text and rating), customised based on the situations that the users just experienced, and then use the form to request users for feedback. We expect that this approach would engage those users that do not have a habit of giving feedback.

Acknowledgements

This research has been conducted within SUPERSEDE project funded by the European Union’s ICT 2014 under grant agreement no 644

Appendix

Table 2-6. Use of feedback tools in different device

	Smart phone use /PC use /no use	**Scenario 1**	**Scenario 2**	**Scenario 3**
Text	Smart Phone	63	64	64
	PC	77	78	74
	total intention to use	98	95	90
	No use	2	5	10
Rating	Smart Phone	51	46	32
	PC	48	45	41
	total intention to use	68	65	57
	No use	32	35	43
Emoji	Smart Phone	47	39	27
	PC	27	27	20
	total intention to use	56	50	38
	No use	44	50	62
Category	Smart Phone	40	35	36
	PC	58	55	49
	total intention to use	70	65	62
	No use	30	35	38
Screenshot	Smart Phone	45	33	35
	PC	64	50	43
	total intention to use	82	64	60
	No use	18	36	40
Audio recording	Smart Phone	35	26	24
	PC	19	23	21
	total intention to use	43	40	35
	No use	57	60	65
Screen recording	Smart Phone	22	22	23
	PC	31	27	29
	total intention to use	45	41	41
	No use	55	59	59
Attachment	Smart Phone	20	17	21
	PC	55	43	36
	total intention to use	64	42	47
	No use	36	48	53

Table 2-7. Descriptive results for analysis of feedback characteristics

Explaining complicated situations

	Min	Max	Mean	Std. Deviation
Text	1	5	4.58	.713
Rating	1	5	2.65	1.403
Emoji	1	5	2.15	1.290
Category	0	5	2.74	1.194
Screenshot	0	5	3.18	1.258
Audio recording	0	5	3.83	1.181
Screen recording	0	5	3.33	1.319
Attachment	0	5	3.69	1.277
Valid N	100			

Tailoring

	Min	Max	Mean	Std. Deviation
Text	3	5	4.60	.620
Rating	1	5	3.69	1.134
Emoji	0	5	3.06	1.286
Category	0	5	3.56	1.113
Screenshot	0	5	3.53	1.226
Audio recording	0	5	3.22	1.353
Screen recording	0	5	3.13	1.338
Attachment	0	5	3.57	1.265
Valid N	100			

Transferring Emotions

	Min	Max	Mean	Std. Deviation
Text	2	5	4.33	.697
Rating	1	5	3.56	1.175
Emoji	1	5	4.20	1.015
Category	0	5	2.93	1.265
Screenshot	0	5	2.55	1.344
Audio recording	0	5	3.60	1.295
Screen recording	0	5	2.61	1.370
Attachment	0	5	2.95	1.282
Valid N	100			

Familiarity

	Min	Max	Mean	Std. Deviation
Text	1	5	4.50	.785
Rating	1	5	4.59	.698
Emoji	0	5	3.39	1.317
Category	1	5	3.93	1.027
Screenshot	0	5	3.20	1.443
Audio recording	0	5	2.28	1.364
Screen recording	0	5	2.29	1.387
Attachment	1	5	3.49	1.210
Valid N	100			

Perceived Usefulness

	Min	Max	Mean	Std. Deviation
Text	2	5	4.54	.717
Rating	2	5	4.32	.815
Emoji	1	5	3.39	1.348
Category	0	5	3.87	1.079
Screenshot	0	5	3.66	1.241
Audio recording	0	5	2.99	1.396
Screen recording	0	5	3.11	1.325
Attachment	0	5	3.66	1.208
Valid N	100			

Perceived Easiness

	Min	Max	Mean	St. Deviation
Text	2	5	4.51	.718
Rating	2	5	4.58	.699
Emoji	0	5	4.22	1.021
Category	0	5	4.07	1.047
Screenshot	0	5	3.64	1.267
Audio recording	0	5	2.95	1.313
Screen recording	0	5	2.97	1.425
Attachment	0	5	3.84	1.152
Valid N	100			

Chapter 3 : The Effect of Requests for User Feedback on Quality of Experience

[The chapter is based on: *F. Fotrousi, S. Fricker, M. Fiedler (2018). "The effect of Requests for User Feedback on Quality of Experience", Software Quality Journal, 26(2), 385-415. https://doi.org/10.1007/s11219-017-9373-7*]

Abstract

Companies are interested in knowing how users experience and perceive their products. Quality of Experience (QoE) is a measurement that is used to assess the degree of delight or annoyance in experiencing a software product. To assess QoE, we have used a feedback tool integrated into a software product to ask users about their QoE ratings and to obtain information about their rationales for good or bad QoEs. It is known that requests for feedback may disturb users; however, little is known about the subjective reasoning behind this disturbance or about whether this disturbance negatively affects the QoE of the software product for which the feedback is sought. In this paper, we present a mixed qualitative-quantitative study with 35 subjects that explores the relationship between feedback requests and QoE. The subjects experienced a requirement-modelling mobile product, which was integrated with a feedback tool. During and at the end of the experience, we collected the users' perceptions of the product and the feedback requests. Based on the users' rational for being disturbed by the feedback requests, such as "early feedback," "interruptive requests," "frequent requests," and "apparently inappropriate content," we modelled feedback requests. The model defines feedback requests using a set of five-tuple variables: “task,” “timing” of the task for issuing the feedback requests, user’s “expertise-phase” with the product, the “frequency” of feedback requests about the task, and the “content” of the feedback request. Configuration of these parameters might drive the participants’ perceived disturbances. We also found that the disturbances generated by triggering user feedback requests have negligible impacts on the QoE of software products. These results imply that software product vendors may trust users’ feedback even when the feedback requests disturb the users.

Keywords

Quality of Experience, QoE, user feedback, user perception, human factors

1. Introduction

User feedback is essential for managing and improving software products (Pagano and Brügge 2013). User feedback informs software companies in identifying user needs, assessing user satisfaction, and detecting quality problems within a system (Fotrousi et al. 2014). User involvement is an effective means for capturing requirements, and, when feedback is considered in decisions about system evolution, it has positive effects on user satisfaction (Kujala 2003).

A well-known indicator for measuring user satisfaction is Quality of Experience (QoE). QoE is defined as “the degree of delight or annoyance of the user of an application or service” (Le Callet et al. 2012). The QoE indicator is sensitive to the fulfilment of user needs. High QoE values reflect users’ enjoyment in using a suitable system (“delight”). Low QoE values reflect users’ dissatisfaction in using an unsuitable system (“annoyance”).

QoE is believed to be affected by three factors: the system, the context in which the system is used, and the software users (Reiter et al. 2014). System factors include the properties and characteristics of a system that reflect its technical quality, such as its performance, usability, and reliability (ISO/IEC 25010). System characteristics reflect the Quality of Service (QoS) of a product (Varela et al. 2014). The context reflects the user environment, which is characterized by physical, social, economical, and technical factors. The users, ultimately, are characterized by rather stable demographic, physical, and mental attributes, as well as more volatile attributes, such as temporary emotional attitudes. When interpreting user feedback, all three factors must be taken into consideration, since all of these factors, and not only the software system, affect human emotions (Barrett et al. 2011).

Some studies have empirically evaluated the impacts of systems, their contexts, and human factors on QoE. Most of these studies have investigated the impact of the system factor, including, particularly, the QoS. For example, Fiedler et al. (2010) investigated a generic relationship between QoS and QoE and presented a mechanism for controlling QoE in telecommunication systems. Other studies have investigated the impact of the human factor (Canale et al. 2014) or the context factor (Ickin et al. 2012) on QoE.

By nature, these impact evaluation studies necessitate frequently asking users for feedback on software products, software features, groups of features, or users' actions (e.g., pressing a button). Especially in QoS-oriented studies, such feedback is necessary to interpret the recorded QoS data (Fotrousi et al. 2014). Automated support for feedback requests enables the quick and easy collection of data from a large number of users (Ivory and Hearst 2001).

However, asking for user feedback may disturb users and introduce bias in their QoEs. Research has shown, for example, that users may be disturbed by badly timed (Adamczyk and Bailey 2004; Bailey et al. 2001) or overly frequent feedback requests (Abelow 1993). While research has objectively investigated the impact of feedback requests on users' annoyance, no work has yet subjectively investigated this issue or explored how users rationalize their annoyance. Furthermore, the extant literature has not yet investigated whether the QoE of the product under evaluation is affected by users' annoyance. As a result, we do not know whether the QoE of a software product may be trusted in cases involving nuisance (Jordan 1998). This uncertainty is particularly important if nuisances are created easily and rapidly.

This paper evaluates whether disturbing feedback requests affect the QoE of a software product. We used a simple probe to collect extensive user feedback, including quantitative QoE ratings and qualitative user rationales. To generate a wide variety of feedback constellations, the probe was triggered randomly as users were implementing a variety of tasks. Some of the users' tasks required little attention, while others required the users to concentrate. The random prompting of different concentration levels for different tasks generated a wide variety of situations in which the users were asked for feedback. At the end of the product usage, a post-questionnaire was administered to collect each user's overall perception of the feedback requests and experience of using the software product. We analysed the collected data to identify the users' rationales for being disturbed by the feedback requests, to determine whether the feedback requests affected the quality judgment of the software product, and to discover whether the feedback mechanism implemented in the probe was used to provide feedback on the feedback requests.

The main contribution of this paper is an understanding of the extent to which disturbing feedback requests affect users' QoEs, which is an area that has been largely overlooked in previous research. Meanwhile, based on users' subjective reasoning for being disturbed by the feedback tool, we propose a feedback request model, which parametrizes the characteristics of the feedback request. Finally, we discover whether feedback tools can be used to capture the disturbances of the feedback requests. The findings in this study will guide researchers and practitioners in designing user feedback mechanisms to collect informative user feedback, which will assist in enhancing software engineering activities, such as requirement engineering, user-based software development, and the validation of software products.

The remainder of the paper is structured as follows: Section 2 provides an overview of the study background and related work. Section 3 describes the research questions, the research methodology, and the threats to validity. Section 4 describes the results and the analysis used to answer the research questions. Section 5 discusses the results. Section 6 summarizes and concludes the paper.

2. Background and Related Work

User feedback reflects information about users' perception of the quality of a software product. Such perceptions can result in positive feelings such as delight, engagement, pleasure, satisfaction, happiness or negative feelings such as disengagement, dissatisfaction, sadness or even combinations of the feelings. The perception differs based on the users' expectations (Szajna and Scamell 1993) in different social contexts (Van der Ham et al. 2014).

User feedback is captured in written, verbal, audio and video formats directly from users or indirectly through the interpretation of users' activities. A questionnaire is an example of methods gathering data in the written format by questioning user feedback. The user feedback can be collected through a long questionnaire (Herzog and Bachman 1981) capturing more data rather than a short questionnaire (Kim et al. 2008b) capturing fewer data but from many users. The short questionnaire can be paper-based or online-based forms. The short questionnaire may also be triggered (Froehlich et al. 2007) regularly or at a particular moment of experiencing a prototype or a released product. The annotating method is another example of written user feedback that users provide comments or rates for snippets of an image (Ames and Naaman 2007) or a video (Fricker et al. 2015) when the users have some opinions to share. The interview (Ahtinen et al. 2009) is an example of methods gathering verbal user feedback. The user feedback can also be recorded in the form of a multimedia such as an audio or a video. User sketch method (Tohidi et al. 2006) is an example of methods for collecting the activity-based user feedback. A user feedback tool includes one or multiple user feedback mechanism(s) implementing one or multiple user feedback methods respectively for collecting user feedback.

The feedback is collected in the form of qualitative or quantitative measures. A qualitative measure provides a verbal and comparative description of the users' opinions. A quantitative measure is a numerical form of data that is usually referred to a number. Mean Opinion Score (MOS) is a known quantitative metric usually scaled ordinal between 5 to 1 (Excellent, good, fair, bad, poor) that subjects assign to their opinion (ITU-T 2003) to measure Quality of Experience (QoE).

Raake and Egger (2014) define Quality of Experience (QoE) as the degree of delight or annoyance of a user, who experiences a software product, service or system. QoE results from the evaluation of the user whether his or her expectations are fulfilled in the light of the user context, and personality. Quality of Experience combines the terms *Quality* and *Experience*. *Quality* is an attribute of a software product that refers to the goodness of the software product. *Experience* is an attribute of the user entity that refers to the stream of users' perception including feelings. Quality of experiencing (QoE), as the combination of the two terms *Quality* and *Experience*, is the user's judgment of the perceived goodness of the software as a cognitive process on top of the experience (Raake and Egger 2014).

Along with the development of a user's experience, the perceived quality of the experience is likely to change over time (Karapanos 2013). During the experience development, the user initially gets familiar with the product and learns the product's functionalities. The user excitement and frustration generated in the familiarization phase may affect the QoE of the software product. However, when the user establishes the functional dependency and is attached emotionally to the software product in the next phases (Karapanos 2013), the judgment of the QoE would be more accurate.

The system, the context, and the human factors may also impact on the judgment of users' perception and affect QoE of a software product (Reiter et al. 2014; Roto et al. 2011). The three factors reflect the reason behind a particular perception of a user in an experience. Context and human factors can determine how the system factors impact on QoE (Reiter et al. 2014). As an example, the same software product may leave different quality perception when is used on a small-size touch screen phone in a car or on a personal computer at home.

The system factors point out to the technical characteristics of a software product or services. The functionality of a software product, delay in data transmission and a content of a media are examples of the system factors. Most of the system factors are relevant to the technical quality of the product or service referring as Quality of Service (QoS). The QoS factors are about the end-to-end service quality (Zhang and Ansari 2011), network quality (Khirman and Henriksen 2002) and suitability of service content (Varela et al. 2014). The QoS factors tends to differ among application domains like: speech communication (Côté and Berger 2014), audio transmission (Feiten et al. 2014), video streaming (Garcia et al. 2014), web browsing (Strohmeier et al. 2014), mobile human-computer interaction (Schleicher et al. 2014), and gaming (Beyer and Möller 2014). As an example in speech communication (Côté and Berger 2014), the quality of the transmitted speech such as loudness, nearness, clearness may affect QoE.

The context factors refer to the user environment characterized by physical, temporal, economical, social, and technical context factors (Reiter et al. 2014). We can exemplify the physical, temporal, social and economical factors respectively by an experience occurs in an indoor or outdoor physical environment, in a certain time of day, based on an individual or a group work experience and with a specific subscription type. The technical context factors are the system factors that contextually related to the software product or service. As an example of the technical context factors, we can mention the characteristics of the feedback tool and a device that the software product has interconnection with, such as the design layout, screen size, and resolution of the device (Mitra et al. 2011).

The human factors characterize demographic, physical nature, mental nature as well as emotional attitudes of human users (Le Callet et al. 2012). The level of expertise and visual acuity of users are examples of demographic and physical factors respectively. Needs, motivations, expectations, moods exemplify the mental factors. Among the human factors, the emotion factor has the strongest relationship with experience (Kujala and Miron-Shatz 2013). For example, the user's frustration in an experience may turn into anger, and the pleasant experience makes the user happy. The users' perception of the product’s quality is influenced by a variety of emotions (Fernández-Dols and Russell 2003). Therefore, emotions are important factors to be considered while studying QoE.

There are studies that have empirically evaluated the impacts of the system, context, and human factors on QoE of a product or service. Fiedler et al. (2010) investigate a generic relationship between system factors and QoE. The authors present a QoE control mechanism, where MOS is a function of QoS metrics such as response time in the telecommunication area. Ickin et al. (2012) investigated the factors that influence QoE in mobile applications. The study findings reveal the effect of context factors such as battery efficiency, phone features, and cost of application or connectivity on QoE. The study also showed the effect of human factors such as user routines and user lifestyle on QoE. Such impact studies are dependent on a frequent automatic collection of user feedback to interpret the quantitative analytics of the system quality that are also automatically collected. Automatic frequent asking for user feedback may disturb users and may bias the judgment of users for QoE of the software product.

We found no work that evaluated whether the request for feedback would affect QoE of a software product. It is quite imaginable that a feedback request would be a part of the system, context and human factors that influence on QoE. Triggering the feedback requests, whose functionality may be perceived as a part of the product (i.e., system factor) interrupts the user’s task. The interruption that occurs in a certain context like mobile context (i.e., context factor) may disturb the user (i.e., human factor) especially when the user perceives performing the task as the primary and providing the feedback as the secondary task (Adamczyk and Bailey 2004). Disturbing the user by the feedback requests prompts user’s perception that causes a sensation or set of sensations toward a negative emotion (Solomon 2008). However, there is a gap in the literature whether the negative emotion caused by the feedback requests would be a factor that influences on users’ perception of the software product quality.

Lack of understanding users’ rationale for being disturbed by the feedback requests and the relations between the feedback requests and QoE of a software product would make the product owner unable to judge the appropriateness of the collected user feedback. In spite of appropriate feedback requests that motivate users to provide rich effective feedback (Broekens et al. 2010), inappropriate feedback requests may bias the collected user feedback that may affect the reliability and robustness of the decisions, which the product owner makes.

3. Research Methodology

3.1. Objectives

The overall objective of this study is to evaluate whether the feedback mechanism affects the feedback obtained about the software product. We aim to determine whether a disturbing feedback request negatively affects users' perceptions of the software product for which the feedback is requested. Therefore, we look for identifying the subjective disturbing aspects of feedback requests during the collection of feedback for a software product. We study whether the interruption is the only disturbing factor and, if not, seek to identify other possible disturbing factors of a feedback request based on users' reasoning. Finally, we seek to discover whether the feedback mechanism that disturbs users is useful for collecting feedback about such disturbances. Feedback about the disturbances informs product owners of the problems that the users have experienced with the implemented feedback mechanism.

We summarize the objectives as follows:

OBJ1: Understanding users' reasoning for being disturbed by feedback requests.

OBJ2: Finding out the extent to which disturbing feedback requests affect users' perceptions of a software product's quality.

OBJ3: Understanding whether user feedback is helpful for understanding the disturbances caused by feedback requests.

3.2. Research Questions

We designed the study to answer the following research questions (RQ1, RQ2, and RQ3), which we mapped to the above objectives (OBJ1, OBJ2, and OBJ3, respectively):

RQ1: How do users rationalize the disturbance of feedback requests?

RQ2: To what extent do disturbing feedback requests affect the QoE of software products?

RQ3: Do users provide feedback about feedback requests?

The overall research efforts help to discover whether the collected user feedback can be trusted even if the users are disturbed by the feedback collection process. The answer to RQ1 determines the aspects of feedback requests that could disturb users. Using these findings, we model feedback requests corresponding to a software product. The model guides the selection of a suitable feedback mechanism to assist researchers and practitioners in collecting unbiased feedback. The answer to RQ2 identifies the relationship between the feedback requests and the users' perceptions of the quality of a software product. The answer to this question helps practitioners ensure that their feedback tools do not influence the quality of users' perceptions of a software product's quality. The answer to RQ3 identifies whether users provide feedback on feedback requests when they are asked to give feedback about the software product. This answer will guide researchers and practitioners in determining whether they can use the user feedback provided for a software product to evaluate the feedback requests generated by the feedback tool.

3.3. Study Design

The study used a mixed qualitative-quantitative research approach, which was designed based on multiple embedded case studies (Yin 2014). Figure 3-1 presents an overview of the study design to address the research questions. For the data collection, a feedback tool was used to request feedback randomly from participants while they were using a software product. At the end of the product's usage, the users' perceptions of the feedback requests and the experiences of using the product were collected through a post-questionnaire. The user feedback about the software product during usage, as well as the user feedback that was provided in the post-questionnaire about the feedback requests and the software product, were analyzed individually to answer the research questions.

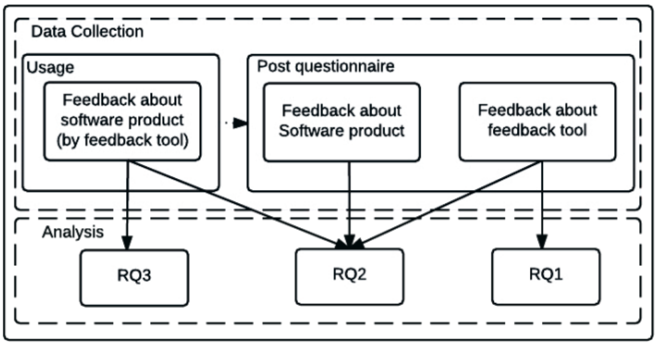
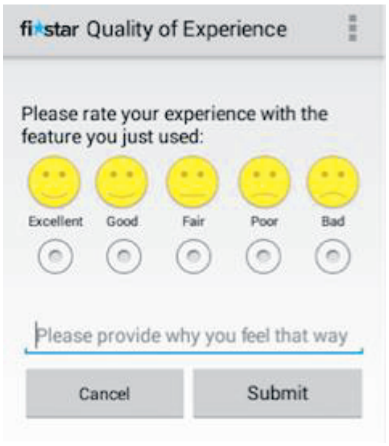

Data Collection
Usage
Feedback about software product (by feedback tool)
Post questionnaire
Feedback about Software product
Feedback about feedback tool
Analysis
RQ3
RQ2
RQ1

Figure 3-1. Overview of the study design

A. Selection of the software product and the feedback tool

As the unit of analysis, we investigated individuals' feedback to determine whether the feedback was about the software product or about the feedback requests. All participants in this study used the same software product and the same feedback tool with the same configuration for requesting feedback.

The QoE probe described by (Fotrousi and Fricker 2016) was used as the feedback tool for collecting QoE data from a requirement-modeling software called Flexisketch (Golaszewski 2013; Wüest et al. 2012; Wüest et al. 2015). We integrated the QoE probe into the Flexisketch tool. Figure 3-2 presents a user interface of the feedback tool. This tool generated requests for feedback continuously and randomly in the middle of users' interactions with a software product. The feedback tool asked participants to rate their experiences with the feature that they had just used and to provide a rationale for their choice. Although the time and frequency of requests could be configured, in this study, a sample configuration was set up that allowed the user feedback to be collected randomly.


fi★star Quality of Experience
Excellent
Good
Fair
Poor
Bad
Please provide why you feel that way
Cancel
Submit

Please rate your experience with the feature you just used:

Figure 3-2. Feedback tool

B. Participants

The participants were 35 software engineering students at the graduate level, who familiar with the concepts of requirement modeling. Attempts were made to achieve as large variations as possible among the participants. The participants varied in age, requirement modeling knowledge and experiences with a requirement-modeling tool.

C. Study procedure

From the perspective of the participants, the primary goal of the assigned task was to evaluate hands-on requirement engineering practices. The participants were free to complete the assigned task at anytime and anywhere that they find suitable within the given deadline of two weeks. The assigned task in the course was not graded; however, if the students could pass the assignment, they were rewarded with better grades in their two other course assignments. The course assignment was not mandatory and the students who were not interested in this assignment, could skip that and choose alternative an assignment to receive the same reward.

The participants, in their roles as requirement engineers, were asked to translate a real-world requirements walkthrough into a requirements model. The participants had to complete their tasks individually by studying the provided workshop video of a Drug Supply Manager solution, analyzing the discussed requirements, and modeling the requirements.

The video was captured from a requirement engineering workshop, where the participants were discussing the issues related to the distribution of drugs to patients. The issues could impact the safety of the patients. A requirement engineer, two pharmacists, a patient representative, a software developer, a solution architect, a medical device expert, and a barcode technology expert were attending the workshop. In the workshop video, the pharmacists, among other participants, were looking for a solution to be able to trace back the drug packages in the supply chain, using a globally unique barcode.

In the current study, all participants received the same task to model the requirements defined during 15 consecutive minutes of the video. The participants could choose any 15 consecutive minutes of video that they intend to model. The desired models were modeling diagrams such as a use-case, activity and class diagram. Each participant could model the requirements using even more than one diagram. The participants were free to choose the modeling type and notations. They were told to ensure that the model specified what the stakeholders had defined during the chosen part of the video.

The participants were asked to draw their models in the Flexisketch tool installed on their touch screen devices. Once they accessed an Android tablet, Android smartphone or a multi-touch screen PC, they needed to install the Flexisketch and QoE probes based on the provided guidelines. Alternatively, they were able to use one of the laboratory's tablets to complete their task. The participants received an instruction document providing all required information.

So, each participant used Flexisketch (i.e., a modeling tool), integrated with the QoE probe (i.e., a feedback tool), to model the requirements extracted from the video workshop. While the participants were modeling the requirements, a QoE questionnaire was automatically triggered by the completion of a feature to ask for user feedback. In the feedback tool, the probability of automatic triggering of the questionnaire was set to 10%.

The user feedback was collected across different features of the modeling tool representing a range of complexities, since complexity is a factor affecting users' concentration and task performance (Zijlstra et al. 1999). For example, "save" is a simple feature with low complexity: a user simply presses a button to save the model. By contrast, the "merge" feature for merging two objects of the model is not straightforward and is categorized as a high-complexity feature. When the participants completed the modeling, they were expected to save the model, export it as an image and then create a short requirement document including this image. The participants were free to complete the assigned task at anytime and anywhere that they find suitable within the given deadline of two weeks.

In the last step, the participants were asked to fill in a paper-based post-questionnaire. The questionnaire included two groups of questions about the modeling tool and the triggered feedback requests.

D. Data collection method

The data collection was performed using open- and closed-ended questions in two steps of the study procedure:

1- During the usage of software product: While the participants were using the requirement modeling tool, the feedback tool was triggered randomly (Figure 3-2) to collect the participants' QoEs (i.e., ratings of their experiences) with the features that they had just used in the modeling tool. The feedback tool also collected the participants' rationales, which justified the ratings.

2- Following usage of the software product: After completing their work with the software (i.e., modeling tool), the participants were asked to answer a paper-based post-questionnaire. In the post-questionnaire, we started with general questions about the users' experiences including whether the participants had previous experience working with Flexisketch, similar requirement-modeling tools and Drug Supply Manager systems. Then, the participants were asked the starting time of the video that they had chosen for modeling as well as the time spent on the modeling tool.

Later, we formulated two questions asking for participant feedback. The first question underlined the disturbance term, as identified in the first research question, to determine users' reasoning for being disturbed. In this question (Q12 in Appendix), we also sought to identify the negative influences of feedback requests on modeling activity disturbances. The second question (Q9 in Appendix), asked for the overall user feedback on the software product. The questions about the feedback requests and the software product were formulated as follows:

-- Feedback requests --

How good was the QoE probe in minimizing the disturbance of your modeling work?
Bad (1), Poor (2), Fair (3), Good (4), Excellent (5)
Please explain why you feel that way: ________________

-- Software Product --

How good was Flexisketch as a tool for modeling requirements?
Bad (1), Poor (2), Fair (3), Good (4), Excellent (5)
Please explain why you feel that way: ________________

To design the two questions, we used a five-point Likert scale, including a mid-point (i.e., Fair (3)), to avoid negative ratings in the absence of a middle point (Garland 1991).

E. Data Analysis Method

The questions RQ1 and RQ3 were answered using a qualitative content analysis approach. To answer RQ2, which is the core research question of this study, we triangulated the analysis using content analysis, pattern matching, and statistical correlation analysis methods. The statistical descriptive analysis was also used to support discussion.

E.1 Content Analysis

The analysis procedure followed inductive and deductive content analysis approaches (Elo and Kyngäs 2008). The inductive approach was conventional, with the objective of coding data freely to generate information, and the deductive approach was based on the use of initial coding categories, which were extracted from the hypothesis, with the possibility of extending the codes (Hsieh and Shannon 2005).

Inductive Content Analysis: Since prior knowledge on the phenomenon was limited, we performed an inductive content analysis to find answers for RQ1 and, partially, RQ2. The study started with the collection of qualitative feedback, which participants provided for the feedback requests (issued by the QoE probe) and the software application (Flexisketch) in the post-questionnaires. The analysis was conducted in the following four steps:

Step 1 – Perform initial coding: Participants' quotes, which referred to their qualitative feedback, were analyzed separately. For each quote, we underlined all terms that could have some relation to reflections of participants' experiences or the impact of the software product on the participants' perceptions. We then read each quote again and wrote down all relevant codes. We repeated the process one by one for all quotes.

Step 2 – Form final codes: We grouped the initial codes to form final codes based on shared characteristics, which put different codes in the same categories. For example, the vocabularies that were synonyms or had the same or similar stems, meanings or relevancies were organized in the same category of codes. Observations in other quotes also assisted in the creation and renaming of the final codes. Such groupings reduced the number of codes and increased our understanding of the phenomenon. As examples, the initial codes of "time-to-time," "every tap," "keep pop up," "too often pop up" and "frequently" all referred to the frequency of the feedback requests; these formed the final code "frequent request."

Step 3 – Form categories: We created categories based on a general overview of the final codes. The categories were formed based on the patterns that we recognized within the quotes and, in some cases, our interpretations of the quotes' meanings (Potter and Levine-Donnerstein 1999). Categories merged into a higher level when the merging made sense. The categories were developed independently by the first and second authors, and the final categories were decided in a joint meeting based on a "chain of evidence" tactic (Yin 2014). The correctness of the categories was later evaluated by the third author. Then, we organized the final categories in a matrix, comprising the connections between the participants' quotes and the categories used by the final codes as elements. As explained in section 4.1, the content analysis concluded the matrix by including three categories: *kind of user perception, consequence of disturbance*, and *characteristics of feedback requests*. *Characteristics of feedback requests* was also divided into the sub-categories of *task, timing, experience phase, frequency,* and *content.*

Step 4 – Perform abstraction: In the last step, based on the extracted categories, we performed an abstraction that led to a generic model. We interpreted and discussed this model based on the quantitative data of the given QoE ratings for the feedback requests and the software product.

Deductive Content Analysis: To answer RQ3, we performed a deductive content analysis. The results of this section could also support RQ2. The research was initiated with the following hypothesis formulation:

H: Participants provide feedback for the feedback requests during their usage.

Then, initial categories of codes were organized. The qualitative feedback that participants provided during the usage was coded to test the hypothesis. The analysis was conducted in the following three steps:

Step 1 – Development of an analysis matrix. We developed a matrix to connect the participants' quotes and the initial categories of codes. The connections were filled with the coding data provided in step 2. We used an unconstrained matrix with the possibility to extend the categories during the data coding. We expected that participants would provide feedback in the categories for *feedback request, software product attributes* and *device attributes.* The first category was defined based on the hypothesis, and the next two categories were factors affecting the QoE of a product, as identified before through the inductive content analysis.

Step 2 – Data coding: We reviewed all comments and coded in relevance to the defined categories in step 1. Although we aimed for an unconstrained matrix, no new categories were recognized during the coding. However, new sub-categories were identified. For example, for the software product attributes, we found a *performance* sub-category as a quality attribute that had not been identified during the inductive content analysis.

Step 3 – Hypothesis testing: The coded matrix was a good tool for easily testing the hypothesis. Exploring the codes identified whether any feedback was available about the feedback requests.

E.2 Pattern Matching

Part of the analysis to answer RQ2 used a pattern-matching analytical technique (Yin 2014). In the pattern matching, a hypothesis to be tested—a so-called predicted pattern—was compared with the observed patterns that were concluded empirically. Section 4.2.1 shows the results of the pattern matching research. We performed the pattern matching in the following four steps:

Step 1 – Formulate hypothesis: We formulated the research hypothesis in alignment with the research question. The research hypothesis is referred to as the predicted pattern during the study. This pattern was formulated as an *if-then* relation, where the *if* statement is the *condition* and the *then* statement is the *outcome*. We used an independent variable design with the "sufficient condition proposition" (Hak and Dul 2009), meaning that the *outcome* of the pattern is always present if the *condition* defined in the proposition is present. Therefore, if alternative patterns in the absence of the condition are confirmed, the hypothesis is disconfirmed. The hypothesis was, thus, formulated as follows:

H-P: The Quality of Experience (QoE) of the software product is always perceived to be bad if the feedback request disturbs the participant.

The *outcome* (i.e., "The Quality of Experience (QoE*)* of the software product is perceived to be bad") was always present if the *condition* (i.e., "if the feedback request disturbs the participant") was present.

Step 2 – Select appropriate cases: To investigate the hypothesis, we look for alternative patterns involving the *outcome* in the predicted pattern (i.e., "the QoE of the software product is perceived to be bad"). The absence of the outcome was the criterion for selecting cases. We chose cases in which the participants rated the QoE of the software product as good and then, from among these selected cases, looked for the presence or absence of the *condition*, as defined in the predicted pattern (i.e., "if the feedback request disturbs the participant").

Step 3 – Observe patterns to test the hypothesis: We observed the conditions in the selected cases and then formulated the observed patterns as the result of this step. We conducted our observation in a matrix with two dimensions for the QoE of the software product and the QoE of the feedback request. We also used the participants' justifications in the qualitative feedback relevant to the selected cases to increase the reliability of the observations.

Step 4 – Formulate test results. This step reported the confirmation or disconfirmation of the hypothesis. If the investigation could show observed patterns in the absence of the condition, it would be sufficient to disconfirm the hypothesis.

E.3 Statistical Analysis

We used a correlation analysis to measure the relationships among the observed variables. As part of RQ2, we used the Pearson and Spearman correlation coefficient methods to investigate the linear and monotonic relationships between the QoE of the software product and the QoE of the feedback request, respectively. Furthermore, throughout the study, descriptive analysis statistics, such as average and median, were used to provide supportive information for the discussion.

3.4. Threats to Validity

Following the classifications in the qualitative study (Yin 2014) and the content analysis (Potter and Levine-Donnerstein 1999), we analysed threats to validity. We also addressed the threats regarding student participation (Carver et al. 2003).

Reliability: We interpreted reliability as the rigor and honesty with which the research has been carried out. Threats to reliability affect the repeatability of the study (i.e., the ability to run the study again and achieve the same results). To address potential threats to reliability, we developed a study protocol, collected all data in a study database, and used triangulation as the main strategy for answering the research questions (Golafshani 2003). We performed data triangulation by collecting data during and after the use of the application and considered both quantitative and qualitative data. We combined quantitative and qualitative approaches for the data analysis. The second and third authors of the study reviewed the results and the analysis performed by the first author.

A key concern was the coding of the collected qualitative user feedback (Potter and Levine-Donnerstein 1999). To mitigate coding problems, the first author documented the design of the content analysis and developed detailed coding rules in a guideline that ensured that the other researchers would make the same decisions when selecting codes. The authors reviewed the coding and discussed conflicting coding results. Inaccurate punctuation and mistyped words sometimes changed the entire meaning and interpretation of a user's feedback. In cases in which the user's intended meaning was unclear, the quote was removed from the analysis.

Internal validity: The threat is the extent to which the results may have been biased by confounding factors. One of the risks in this study was that the users might be disturbed by another stimulus, such as their devices or the physical environment, rather than by feedback requests. We captured the causes for such disturbances using the qualitative feedback received from the users during and after their experiences with the software product. Capturing these factors assisted us in distinguishing them during the analysis.

One factor that could have biased the entirety of the study results was the participation of students. The participating students could have felt incentivized to provide the results that their teacher(s) expected. To mitigate this threat, the first author, who executed the study, was not involved in the teaching of the concerned course. In addition, the assignment was optional for the students and not graded. The participants could voluntarily select either this assignment or another alternative assignment of comparable effort and difficulty. The participants could also opt-out at any moment and choose to do another assignment.

Another potential confounding factor related to insufficient information for the participants, which could affect users' disturbance. To mitigate this threat, we informed the participants that the task was part of a research project and explained the roles of the QoE probe and the Flexisketch. The participants also had access to the post-questionnaire in advance. Furthermore, we informed the participants about the monitoring of their usage data, which would be kept anonymous. Such monitoring data could be used to enhance internal validity and, to some extent, replaces the actual observation of the participants as they performed their tasks.

External validity: External validity concerns the ability to generalize the results obtained from a study. In this study, fourth-year software engineering students participated as subjects. They did not have knowledge of user feedback research, but they had been introduced and extensively trained in software engineering, including in theory and team projects. In a comparable rating and feedback study, Fricker et al. (2015) could not identify discernible differences between student ratings and ratings of industry subjects and noted that their positive and negative feedback were congruent. Similarly, Höst et al. (2000) could observe only minor differences in the conception, correctness, and judgment abilities of last-year students and professionals. Not only the number of analysis units (i.e., user feedback) but also the number and kind of case (i.e., modeling of Drug Supply Management requirements) are important for generalizability.

The findings contribute towards generalization as they are applicable to the cases with similar characteristics. For instance, the findings can be applied to the cases where the users require a high level of creativity and interaction with the software (e.g., Adobe Photoshop modelling software) to perform their tasks. However, as Kennedy (1979) recommends for a single case, we leave the judgment for generalizability of the case to the practitioners, who wish to apply the findings, to determine whether the study's case is applied to their own case. In the end, to corroborate further generalization of the research results to other settings, similar research studies with other types of subjects and different software products should be conducted.

Construct validity: Construct validity reflects whether a study measures what was supposed to be measured. The risk in this research was that the participants might provide feedback without really experiencing the requirements modelling product or that, in the event of this experience, they might not provide sufficient evidence in their feedback to answer the research questions. To mitigate the threat of students providing feedback without experiencing the product, the study protocol forced the participants to report the results they had achieved with the software product. In this protocol, we also established a chain of evidence to ensure that the categories were defined correctly during the content analysis. We also reported the analysis by making explicit (e.g., by reporting quotes at appropriate places) how our answers to the research questions were based on the data we collected.

Furthermore, in real environment users could perform such tasks within few hours. However, the time pressure on the participants for performing their tasks could be a risk that might result in reducing the quality of the answers (Sjøberg et al. 2003). The time pressure might make the participants more anxious and lead different judgment (Maule et al. 2000) on the given user feedback. To reduce the threats to validity, the design of our study allowed the participants to perform their task in a relax time within two weeks.

The complexity of tasks is another threat to construct validity as different complexity might cause a different level of concentration and task performance (Zijlstra et al. 1999). Therefore, we considered several variations in our design to cover a wide spectrum of complexities from low-complexity (e.g., pressing a button, or watching a simple and understandable video), to high-complexity (e.g., merging two objects) tasks.

4. Results and Analysis

The results show that the 35 study participants were from Europe (42.9%), China (42.9%), Africa (8.6%), and the Middle East (5.7%). Of the participants, 22.9% were female, and 77.1% were male. All were aged 23 to 37 years old, with the mean of 25.7 years. Table 3-1 gives an overview.

Table 3-1. Distribution of participants: country (left) and gender (right).

	Country					**Gender**		
	Africa	China	Europe	Mid-East	Total	Male	Female	Total
Frequency	3	15	15	2	35	8	27	35
Percentage	8.6	42.9	42.9	5.7	100.0	22.9	77.1	100.0

None of the participants had previously experienced the requirement modeling tool and Supply Manager applications. To conduct the task, the participants used several models of Android tablet, Android smartphone, and no use of a multi-touch screen PC was reported. They participants reported their duration of using the requirement modeling tool. The responses ranged from two hours to four days. From the answers collected during the post-questionnaire, the participants rated the feedback requests and the software product in the range of Good (4) to Bad (1), with a median of Fair (3). No Excellent (5) rating was collected.

Table 3-2 shows the number of submitted feedback on the software product or feedback tool. According to the usage log, 25 participants provided feedback on the software product during runtime. Although the 10 remaining participants had seen the feedback tool at least 2 times while performing their task, but they did not submit any feedback, i.e., They declined the feedback requests. Based on the instructions given to the participants, the participants were able to decrease the likelihood of triggered feedback requests or deactivate the feedback requests. Ten user feedback less on software products means we missed some qualitative feedback at the feature level, which was not critical of our analysis.

Table 3-2. Number of submitted feedback.

	Feedback on software product (Usage log)		Feedback on software product (Post-questionnaire)		Feedback on Feedback tool (Post-questionnaire)	
Usage log	QoE rating	Rationale	QoE rating	Rationale	QoE rating	Rationale
Participants	25	25	35	33	35	33
Total feedback	441	60	35	33	35	33

The participants submitted a total of 441 QoE ratings and 60 valid feedback that justified these ratings during product usage (64 feedback rationales were provided, which four were made of meaningless letters or symbols). The QoE ratings were distributed in the range of Excellent (5) to Bad (1) (i.e., Excellent (5): 70, Good (4): 133, Fair (3): 77, Poor (2): 89, Bad (1): 72 feedback). The users provided rationales when they had both positive and negative perception (i.e., Excellent (5): 7, Good (4): 13, Fair (3): 8, Poor (2): 22, Bad (1): 10 feedback). The Median of QoE ratings with Rationale and without Rationale (i.e., Poor (2) and Fair (3) respectively) shows that the participants have more justified the feedback ratings when they had a negative perception.

All participants returned the post-questionnaire. 33 provided rationales for the ratings, while two did not. Figure 3-3 gives an overview of the QoE ratings of the software product and the QoE rating of the feedback requests that have been collected from the post-questionnaire. As presented in the top-left chart, the perceived quality of the feedback requests was less than the perceived quality of the software product. Since the scale defined for the QoE rating was the Opinion Score, an ordinal scale, we calculated the Median as the measure of central tendency: $\text{Median}_{\text{QoE of feedback requests}} = 2$, $\text{Median}_{\text{QoE of software product}} = 3$, equivalent to the Poor (2) and Fair (3) levels, respectively. The levels show that the participants were disturbed by the feedback requests. The software product was appreciated better, even though clearly not excellent. According to 5-point Likert scale used in designing questionnaires as well as our non-parametric statistical test, levels 2 and 3 are significantly different. The level 2 refers to unsatisfactory perception, while 3 shows the mid-point referring to uncertain perception.

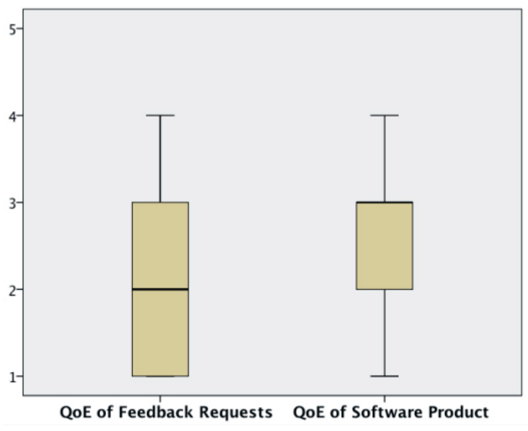
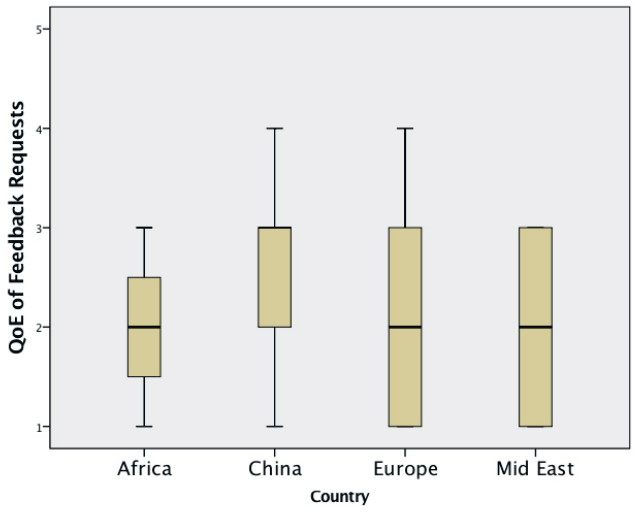


Figure 3-3. Distribution of the participants' ratings for the QoE of the feedback tool and the QoE of the software product according to the post questionnaire*

*: The QoE scales reflect the Opinion Score from Bad (1) to Excellent (5).

Figure 3-4 shows an analysis of the influence of cultural diversity on QoE. For the majority of countries $\text{Median}_{\text{QoE of feedback requests}}$ = 2. The Chinese participants differed with a median QoE rating of 3. In addition, $\text{Median}_{\text{QoE of software product}}$ = 3, except for the participants from Middle-East, who rated the software product to be Good (4). No country reversed the results shown in Figure 3-4, suggesting that cultural differences had no effects that would reverse the study results. The participants were disturbed more by the feedback requests than by the software product, and the Good (4) ratings were likely due to the small number of participants.



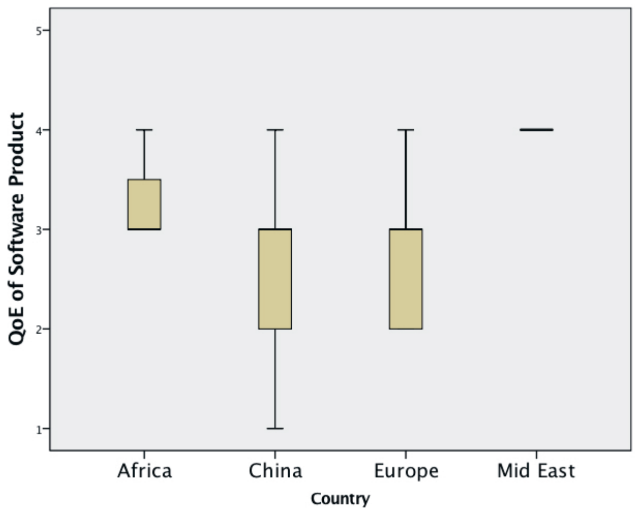


Figure 3-4. Distribution of the participants' ratings for the QoE of the feedback tool (top) and the QoE of the software product(bottom)*

*: According to the post questionnaire. The QoE scales reflect the Opinion Score from Bad (1) to Excellent (5).

4.1. Modelling of Feedback Requests

Based on the qualitative analysis below, we modelled a feedback request according to the users' reasoning for the disturbance level of the feedback tool. As presented in Equation 1, our model defines a set of feedback requests for each product (p) and user (u). Each product (p) and user (u) belongs to the set of available products (P), respectively users (U). The FRs are a set of five-tuple variables referring to the user task (ta), the timing of the feedback request within a task (ti), experience-phase (e), the frequency (f) of the feedback request, and the content (c) of the feedback request.

$$FR = \{(ta, ti, e, f, c) \mid p \in P, u \in U\}$$

Equation 1. Model for user feedback requests developed from the inductive content analysis.

The *user's task (ta)* refers to the type of activity the user was performing with the software product when a feedback request was issued. The important user's tasks were modelling requirements and managing the model, e.g., by saving it. The *timing (ti)* is the moment within the user's tasks when the feedback request has been issued. The *expertise-phase (e)* refers to the user's stage of understanding and mastery of the product at the moment of the feedback request. For example, in a modelling tool, the experience-phase can refer to the learning period at the beginning of an experience. The *frequency (f)* of a feedback request refers to the maximum number of times that feedback is requested in a specific timing and expertise-phase relevant to the task. The *content (c)* refers to the questions included in a feedback request. The values for any of these variables might drive the perceived disturbances.

The feedback request model is a result of the inductive content analysis described in section 3.3.E1. During the content analysis, we identified that the participants' quotes referred to three main categories: *kind of user perception, consequence of disturbance*, and *characteristics of feedback requests*. *Characteristics of feedback requests* could be divided into the sub-categories of *task, timing, experience phase, frequency,* and *content.* Each of the variables ta, ti, e, f, and c reflect one of these identified categories.

The categories were identified based on the users' subjective reasoning for disturbing feedback requests. The following disturbing issues were identified:

- a feedback request that was interrupting a user task;
- a feedback request that was issued to the user too early before the user experienced enough and understood the product;
- a feedback request that was issued too frequently; and
- a feedback request with apparently inappropriate content.

The first three factors were mapped to the timing within a task, the expertise-phase, and the frequency of the request for the task. The fourth factor concerned the content of the feedback request and the functionality provided to allow the user to give feedback. In the following, we show the users' reasoning for the disturbance of feedback requests. These are supported by the participants' quotes (written in italic fonts within quotation marks) to improve the credibility of the discussion.

The participants perceived that the tasks were interrupted at the macro, meso, and micro levels. The participants provided their rationales for being disturbed in macro-level (e.g., modeling), meso-level (e.g., drawing diagrams or working with features, such as locating UML elements), and micro-level (e.g., performing an action, such as a click). Although the interruption was generated at the meso-level (end of using features), however, some participants perceived the interruption in the micro-level. We argue that this incorrect perception could be due to less than a second delay of showing the feedback form. Also, another reason could be due to fragmentary user's action, where the system recognizes it as the end of using the feature (e.g., releasing the mouse button in the middle of drawing a line that the system identifies a new line). The interruption was more disturbing when the task required concentration.

> "… Let me put an example, if I want to put down a square, add a text and put the text in the square, then I don't want to be disturbed while doing that. I don't mind if QoE Probe disturbs me after I've done this

> few concatenated steps, but this was not the case. It kept interrupting me ...”
> “… sometimes you could lose a bit track of a thought process and when that happened it was quite annoying …”
> “It was annoying as it asked while I was drawing and then only half the line was finished.”

A feedback request that came too early before the user had the chance to really understand the product disturbed participants. Because the user expertise of whom received early feedback requests was still in the learning phase and familiarization with the product. In response to an early feedback request, a participant was unable to judge a product, feature, or action, and the judgment risked not reflect sufficiently complete, accurate, or correct feedback.

> “I think it should leave at least a week for users to experience the app[lication], then they will have a better understand and experience of the Flexisketch.”

The frequent feedback that was requested at multiple times during a task disturbed users. Frequent requests increased perceptions of disturbance when the same feedback requests were repeatedly asked for the same feature or action. Sometimes, the feedback request was issued so frequently that the participants perceived that the main goal of the study was to disturb them.

> “Way too intrusive as it came up way too often.”
> “I had to write feedback multiple times for some features, while for others – never.”
> “It felt as if the entire purpose of the QoE Probe was to disturb my modeling work.”

The feedback that was requested frequently encouraged participants to discover the mechanism behind triggering the feedback questionnaire. Due to the ambiguity of this mechanism, the users could even lose sight of the main objective of the feedback requests.

> "It was really disturbing, it disappears after a while, but again I don't know it was on me or the system that solved it."
> "To be honest I do not know why I need to install it."

The content of a feedback request was also mentioned as a disturbing factor, although its impact level (relevant to participants' ratings) was not considerable. The participants complained that the feedback requests had limited functionalities.

> " The function [of feedback requests] is quite limited…"
> " … the functions [of feedback requests] are not as good as I wished. "

Not only did unsuitable feedback requests disturb the participants, but the participants also expressed feelings of annoyance and disengagement.

> "The interruptions were too many and not welcome."

Such feelings consequently affected the quality of the provided feedback and the quality of the participants' performance on the main tasks in the experience. Disturbed participants might be discouraged regarding the provision of feedback, or they might provide inaccurate feedback. Furthermore, participants' task performance was reduced when the participants lost track of their thoughts and forgot their next tasks due to the interruptions. Such disturbances encouraged participants to take action, such as uninstalling the feedback tool.

> "Since it pops up in the middle of working on a diagram, you don't have much will and time to think truly carefully before answering. This probably means that the results aren't as accurate as one could wish for."
> "…I felt it disturbing most when the QoE came up in the middle of me having an idea I needed to model. By the end of my feedback, I almost forgot what I was about to model, which was for me very annoying. …"

"it disturbed my modeling quite a lot I was almost tempted to uninstall it."

The majority of participants who mentioned higher levels of disturbance or efforts to take give-up actions, such as uninstalling the feedback tool in their quotes, rated the QoE of the feedback tool as a 1 or a 2. However, the participants rated the QoE of the feedback tool as a 3 or a 4 when they did not recall a high disturbance level; instead, these participants used occasional adjectives, such as "some" or "sometimes," to describe their disturbances due to frequent/interruptive feedback requests.

4.2. The Effect of Disturbing Feedback Requests on the QoE of a Software Product

Disturbing feedback requests have a negligible impact on participants' perceptions of the quality of software products. The QoE of a software product does not correlate with the disturbance ratings of the feedback requests. The results show that the QoE of a software product might not be degraded even by participant feelings of disturbance related to the feedback requests. Even though the feedback request characteristics discussed in Section 4.1 might disturb the participants, the quality of the software (i.e., 97% of the quotes) and the context such as the device quality (i.e., 42% of the quotes) served as the focal points of arguments to justify the QoE ratings.

The study's results were triangulated with three individual analysis methods to facilitate studying the phenomenon from different angles. This section details these analyses.

4.2.1. Was the QoE of the software product bad when the feedback request disturbed participants?

A disturbing feedback request did not necessarily indicate that participants would negatively evaluate the QoE of the software product. In other words, the disturbances caused by the feedback requests did not always result in a bad experience of the software product. This statement was concluded as the result of disconfirming the predicted pattern we identified for this study, as follows:

P: The Quality of Experience (QoE) of the software product is always perceived to be bad if the feedback request disturbs the participant.

The analysis showed that the QoE of the software product was perceived to be good even when the feedback requests disturbed the participants. As explained in E.2 in Section 3.3, to test the pattern P, we explored the following two possible alternative patterns within the participants' quotes.

AP1: The Quality of Experience (QoE) of software product is perceived to be good, if the feedback request disturbs the participant

AP2: The Quality of Experience (QoE) of software product is perceived to be good if the feedback request does not disturb the participant

We evaluated the alternative patterns AP1 and AP2 using the participants' ratings that were collected via the post-questionnaire and the feedback tool after and during the usage respectively. Figure 3-5 presents the participants' ratings for the feedback requests and the QoE of the software product, collected from the post-questionnaire. The x-axis indicates the ratings of the feedback requests, and the y-axis shows the quality ratings for the software product.

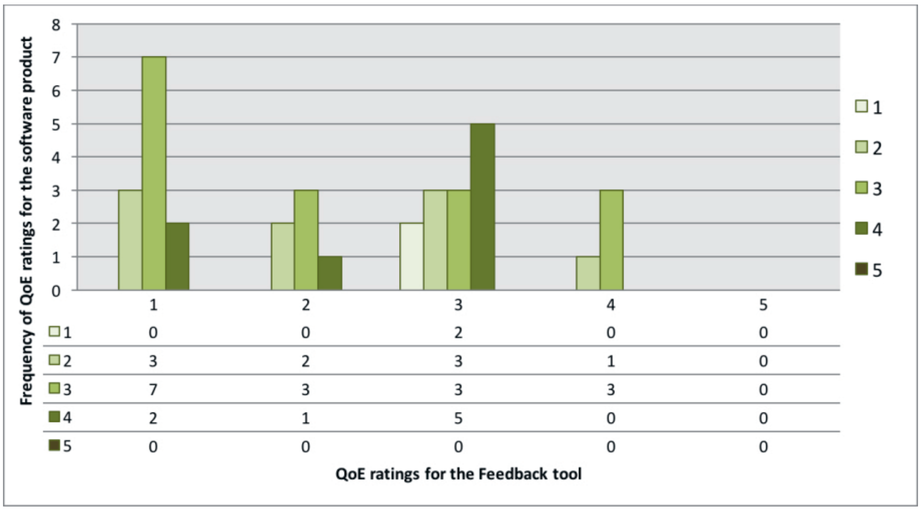
The observation of the alternative patterns *AP1* and *AP2* in the matrix in Figure 3-5 showed that when the QoE of a software product was rated Good (4) (there were no Excellent (5) ratings), in 37% of the cases, the feedback requests disturbed the participants (i.e., rated Bad (1) and Poor (2)); these results aligned with *AP1*. In the same scenario of QoE rating, 63% of the feedback requests did not disturb the participants (rated Fair (3) and Good (4)); these results aligned with *AP2*. The observation of *AP1* contradicted the predicted pattern and, thus, disconfirmed it.

The similar observation was also found in the participants' qualitative motivations. For example, one participant liked the product and rated as a 4 with this rationale:

> "It was fun in creating the diagrams because I was lying on my bed and creating the diagrams by using it. I like it."

However, the same participant was disturbed by the feedback requests, rating these as a 1, with the following rationale:

> "I was just fed up from this QoE because it was disturbing a lot while making diagrams."



	1	2	3	4	5
□1	0	0	2	0	0
□2	3	2	3	1	0
□3	7	3	3	3	0
■4	2	1	5	0	0
■5	0	0	0	0	0

Figure 3-5. Distribution of the QoE of the software product per each QoE of the feedback request (data series reflect the QoE of the software product) - Data is collected via the post-questionnaire.

The pattern *AP1* could also be seen within the feedback collected from the feedback tool. There was one case in which the QoE of the software product was perceived as Excellent (5), but the participant complained about the disturbing feedback requests. The observation of *API* disconfirmed the *P1*.

The examples and the descriptive statistics showed that a disturbing feedback request did not necessarily imply a bad QoE of the evaluated software product.

4.2.2. *Was the QoE of the software product statistically related to the QoE of the feedback requests?*

With the provided ratings, we could not find any evidence to show a dependency between the quality ratings of the disturbing feedback request and the software product.

A correlation analysis was performed to measure the relationship between the participants' ratings given to the feedback request and the quality of the software product, as collected through the post-questionnaires. The results showed a very small, almost non-existent correlation (i.e., Pearson analysis [= -.056, n = 35, p > .001] and Spearman analysis [= -.032, n = 35, p > .001]). The analyses indicated a lack of linear and monotonic relationships between the participant ratings for the quality perception of the feedback request and the quality perception of the software product.

4.2.3. *Were the QoEs of the software product justified with arguments about disturbing feedback requests?*

The QoEs of the software product were justified with arguments about factors other than the disturbing feedback requests. The software characteristics and the experiencing context were the focal points of these arguments.

The participants also provided arguments about the quality of the software product and the experiencing context (e.g., device characteristics) that respectively addressed 97% and 42% of all feedback for justifying the QoE of a software product in the post-questionnaire. Among this feedback, no participant used any characteristics of a feedback request to justify poor QoE ratings for a software product. We could argue that the two separate questionnaires at the end of usage—one for the QoE of the feedback requests and one for the QoE of the software product—allowed the participants to distinguish the feedback tool from the software product. Therefore, the participants provided justifications for the QoE ratings of the software product regardless of the ratings they had given for the feedback requests.

However, the feedback collected by the tool during the usage could not provide enough evidence to justify the QoE ratings. Although four feedback quotes out of 64 were related to the feedback requests, these quotes did not include interpretations of the QoE ratings. For example, one participant, who complained about the interruptions of feedback request two times, gave Poor (2) and Excellent (5) ratings to the QoEs of the same feature.

Software quality attributes were the most common factors that the participants used to justify their ratings. *Functionality, usability*, *learnability, portability,* and *performance* were the quality attributes that the participants most commonly used for these justifications.

Functionality and usability of software features were the most common categories of feedback. Interestingly, of the 33 rationales provided for rating the software product in the post-questionnaire, 19 feedback rationales addressed the software's functionality and 16 feedback rationales addressed its usability categories. Furthermore, out of 60 total feedback quotes, the feedback tool collected 36 and 16 feedback quotes about the functionality and usability categories, respectively.

The participants gave feedback about crashes and errors in product functionality. The participants were also disturbed by nonconformities with the expected functionality. They reported issues with some features that did not work properly or were not successful in fulfilling their expectations.

> "…The zoom function did not zoom text as I wanted, making the model very wired, and the lines which I draw between actor/stakeholder to circles did not connect properly, annoying me as well."
> "Flexisketch seems to lack the following [functionalities]: Arrow heads for directions, copy and, paste mechanisms, screen resize functionality, Eraser functionality, Scrollbar functionality, code generation functionality,.."
> "Because the poor functionalities, and strong dependence on the device (for now it can only run in android system) that don't flexible for the user."

The participants provided feedback on the usability of features, particularly with regard to their ease or difficulty of use. Some of the participants failed to recognize the software product as user-friendly, while others admired its simplicity.

> "It was okay as it had all of the features as you need, but it wasn't user-friendly at all at least not on my phone…."
> "It's fair because the application is very simple and easy to use, but it also has many limitations."
> "The program was literally unusable in horizontal view which was a huge set-back on my smartphone. Some options disappeared while being in horizontal view."

The participants also provided feedback on the performance of the product in relation to an overly long response time.

> "The response is too slow."
> "It takes some time but maybe because of the touch screen quality."

Even when the participants watched or read the instruction guidelines, they still faced learnability issues.

> "I watched the instruction video, but I still don't know how to draw specific items, like arrows."

From the participants' points of view, the context was perceived to be a part of product attributes. The participants also provided arguments about device attributes (e.g., mobility characteristics, screen size, touch-based functionality, and the operating system of the device) as context factors to justify their ratings. The participants complained about using the product on small-sized screens.

> "I think it is useful when I watch the tutorials, but when I really use it, I found it is really not suitable for mobile phone."
> "Too less kind of elements can be chosen to draw a diagram. Not easy to use on a small-screen mobile device."

> "Because the poor functionalities, and strong dependence on the device (for now it can only run in android system) that don't [make it] flexible for the user."
> "This app can be installed in mobile with Android system, which is easy to carry and edit."

4.3. Feedback About Feedback Requests

Of the 64 feedback collected by the feedback tool, only four feedback rationales from two participants, representing 6% of the total qualitative feedback, concerned the feedback requests. The four feedback rationales represented only 0.9% of the total participant experience ratings. Most of the participants did not provide qualitative data (85%); instead, they only rated their experiences.

A few participants gave feedback concerning disturbing feedback requests. Experience interruptions and inappropriate question timing were two categories of disturbing feedback that the participants mentioned.

> "Do not interrupt during drawing!", "This forum really disturbs."
> "Because I am getting the rating without even getting a chance to finish my sketch","The same as a previous comment."

Exploring all of the ratings and the feedback revealed that a majority of participants did not provide qualitative feedback; however, those that did provide such feedback primarily pointed to the quality of the software and the context (as discussed in Section 4.2.3). The feedback was provided both to complain about and to admire the quality of the software product. However, the feedback about the feedback requests was only issued in the case of disturbance. When no issue was found, the participants did not admire the feedback requests.

Although the majority of the participants did not offer feedback on the feedback requests, the few received feedback was still useful for obtaining an accurate understanding of the problems that the participants experienced with the feedback tool.

5. Discussion

According to the findings of our study, feedback requests that are interrupting a user's task, that are too early for what a user knows about the product, that are too frequent, or that are with inappropriate content may disturb users. The first factor is congruent with earlier research. The second and third factors are not surprising, although previous studies did not address them as the disturbing factors caused by feedback requests, but the latest factor is new.

A request for feedback that interrupts a user during a task affects the user's task and, as a consequence, the user's experience negatively (Bailey et al. 2001). In our study, such interruption was particularly problematic during a modelling task, which required particular attention. The interruption generated frustration because the user has to remember the task and how to proceed toward completion of the task. As suggested by Adamczyk and Bailey (2004), it is crucial to find the best moment of interruption and thereby reduce the extent of disturbance.

A feedback request that is issued to a user before he is familiar with the product is perceived to be disturbing. Such familiarization phase is important as a user needs to establish knowledge of the product and how the product is to be used. Some users do not accept the product initially, but they have better perception over prolonged use (Karapanos 2013). Also, the familiarization is accompanied by a change of thoughts, feelings, and expectations about the product (Karapanos 2013). An initially positive judgment of a product may become negative, or vice-versa. Thus, when confronted with a feedback request that is too early, the user may be unable to judge the product or may give feedback that is incorrect. According to our results, the knowledge about this inability if felt by the user as a disturbance. It is important to match the timing of a feedback request with the user's knowledge about what the request is seeking feedback for.

A rapid re-occurrence of requests for feedback disturbs users. This insight is interesting because it extends the understanding of how temporal aspects of feedback requests affect the product user. Even well-timed requests for feedback may be disturbing if they are issued too frequently. Especially disturbing is the repetition of requests if the user has already submitted feedback that was well thought through and well formulated. It is a must for a feedback mechanism to consider the history of the feedback dialogue with a user.

New is that a feedback request that offers too limited functionality in the eyes of the user can disturb as well. This insight is interesting because related work has focused on the aspect of timing feedback requests. According to our data, it is also important that the feedback request gives the user the ability to provide feedback in a way that is intuitive and desired by the user. Our chosen combination of a Quality of Experience rating and a text field for user feedback was perceived to be too limited by some users. Additional capabilities may be needed, such as screenshots, voice, video recordings, or photographs (Seyff et al. 2011).

It is interesting to compare these results with the Qualinet definition of QoE (Le Callet et al. 2012) that we apply here for a feedback tool. According to that definition "Quality of Experience (QoE) is the degree of *delight or annoyance* of a person whose experiencing involves an *application, service, or system*. It results from the person's evaluation of the fulfilment of his or her *expectations and needs* with respect to the *utility and/or enjoyment* in the light of the person's *context, personality, and current state*". A feedback tool *annoys* users if the parameters are not configured well. Users may feel *delight*ed while giving feedback if the feedback has strong *utility*, such as the anticipated improvement of the product in a future release. The study has shown that the *expectations and needs* of the feedback tool are about the timing and content parameters that should be respected when issuing a feedback form. The user's *context, personality, and current state* are reflected in the user's expertise of using the product. We could not identify any other factors in the presented study, including cultural background, that would affect the QoE of the feedback tool.

A feedback request that is disturbing causes negative emotions such as anger (Scherer 2005; Solomon 2008). Such emotions are visible in bad QoE ratings (Antons et al. 2014). The disturbances may also hinder sustained adoption of a product. A user may resist incorporating a product into his daily routines where usefulness and long-term usability are important (Karapanos 2013). Even though the software product may evoke positive emotions in a user, the negative emotions caused by the disturbance may prevent or delay development of emotional attachment to the product. Hence, in addition to offering an attractive product, it is important to present feedback requests satisfactorily, or to offer the possibility to disable the feedback tool.

While feedback may disturb a product's users, our study showed that this disturbance has a negligible impact on the users' reported Quality of Experience for the software product. The users differentiated between a feedback tool they were providing feedback with and the software they were providing feedback for. The disturbance of a user was hardly reflected in that user's QoE ratings for the product. As we could not find any prior study that investigated this perceived separation between product and feedback tool, we believe that this is an interesting new result. The negligible impact implies that software product vendors may trust the collected feedback even if the feedback requests disturb the users to some extent.

In contrast to the perceived separation of a feedback tool and a software product, users blurred the boundary between the software product and the device on which the product was running. The user feedback mixed product and device factors. Perhaps the users could not distinguish the device and the product, or they considered the device to be a part of the product. Thus, a software vendor can receive informative feedback not only about the software product but also the devices the customers are using to run the product on.

Although disturbing feedback requests did not show any significant impact on QoE of the studied software product, the disturbances might affect how well feedback requests are answered. Disturbances may demotivate users to provide rich feedback since the users would ignore disturbing feedback requests. This reaction was evident in that many study participants cancelled feedback requests or switched the feedback tool off. The design of a feedback mechanism is possible through configuring the parameters of the feedback requests model.

The above findings were achieved in a case study that was set up the environment close to reality with less pressure and control on the participants. A pressurized and controlled environment, on one side, could increase the sensitivity of users in response to the environment that might impact on users' perception. On the other hand, such controlled situation could not affect the ability of users to evaluate the software or feedback. Putting users on a regime such as time pressure could amplify the anxiety leading to different judgment (Maule et al. 2000).

Like any other study, also the here presented study has its limitations. For example, we did not research when users decide to decline feedback requests (e.g., cancelling feedback forms). The research could be interesting to investigate the consequence of being disturbed by the feedback requests in a future study. However, this limitation did not affect the presented result in Figure 3-3 that was achieved based on the post-questionnaire. Furthermore, approaches need to be evaluated for including the identified parameters of the user task, feedback request timing, expertise-phase, feedback request frequency, and feedback request content in the design of a feedback mechanism. Finally, users may have different thresholds for feeling affected by disturbance; depending on the situation, some are rapidly disturbed, while others can accept a lot of annoyance (Van der Ham et al. 2014). Therefore, categories of users, contexts, and products may need to be identified to allow investigation of feedback request parameters in each cluster separately. Such research will be future work.

6. Conclusion

Quality of Experience (QoE) is a measurement that is widely used to assess users' perceptions when experiencing a software product. With knowledge about QoE, companies hope to make appropriate decisions to win and retain customers by evolving their products in meaningful ways. Collecting users' QoEs requires automatic and frequent requests for feedback. However, automated requests for feedback may disturb users and perhaps degrade their QoE ratings.

The current study investigated the candidate relationship between the characteristics of automatic feedback requests and the QoE of a software product. The study followed a mixed qualitative-quantitative research method with 35 software engineering participants. We integrated a feedback tool into a mobile software product to prompt participants for feedback randomly in the middle of their experiences. At the end of the users' experiences, we collected their perceptions about the feedback requests and their experiences of using the application through a post-questionnaire.

We offer two contributions to the researcher and practitioner communities. First, we propose a feedback request model that parameterizes the characteristics of feedback requests. The parameters outline the task, timing of the task for issuing the feedback requests, user's expertise-phase with the product, the frequency of feedback requests about the task, and the content of the feedback request. The findings may inform researchers of the parameters that disrupt users' experiences, which may help them develop suitable feedback mechanisms to control users' disturbance. The findings may also help practitioners design the feedback tool and the corresponding feedback mechanisms by adjusting the parameters.

Second, the study showed that feedback requests have negligible impacts on users' QoEs of a software product. Specifically, the quality of the software product has a greater impact on the QoE than the characteristics of the feedback request. For practitioners, this finding implies an ability to trust feedback collected from users, even when the requests for feedback are considered disturbing. The results also imply that the quality of a software product is the most important aspect for practitioners to focus on when examining user feedback. However, the design of suitable feedback mechanisms should not be neglected, since feedback mechanisms are useful for collecting informative user feedback about both software products and any disturbances caused by feedback requests. The informative user feedback assists in enhancing software engineering activities. An informative user feedback assists requirement engineers to elicit new requirements and revise the current requirements for next releases of the software product (Carreño and Winbladh 2013). Such rich feedback also contains valuable information for developers to redevelop a functionality and validate the software product idea (Kujala 1 2008) toward the software evolution (Pagano and Brügge 2013).

The result was achieved based on constructing one situation. However, case variations in practice might stimulate users' emotions differently and lead to new achievements. Therefore, it would be interesting to replicate the study considering several varieties of contextual and system factors in future. The materials for replication is available in http://bit.ly/2o89rO4.

Appendix

Post-questionnaire: Table 3-3 shows the questions that were used to collect user feedback after the participants performed modeling of the requirements.

Table 3-3. Post-questionnaire

=== About Yourself ===

Q1: Your Student ID:

Q2: Did you use Flexisketch already before this assignment? [Yes/No]

Q3: Do you have experience with applications like the Drug Supply Manager (DSM)? [Yes (describe your experience:/No]

=== How You Did the Assignment ===

Q4: Did you install the QoE Probe before doing the modelling with Flexisketch? [Yes/No]

Q5: What device did you use for modelling with Flexisketch (e.g. I would have done it on "Sony Z2 Tablet" – you might have used another one)?

Q6: When in the workshop video did you start the modeling (e.g. "12 minutes after start")?

Q7: When in real-world time did you start the modeling (e.g. "December 14 at 09:04")?

Q8: How much time did you spend for the modelling?

=== Your Experience of modelling with Flexisketch ===

Note again: no grades are given for this assignment – please be honest

Q9: How good was Flexisketch as a tool for modelling requirements? [Opinion Score Scale, rationale]

Q10: Being a potential requirements engineer, would you use Flexisketch again for requirements modelling? [Yes/No, rationale]

=== QoE Probe ===

Q11: Approximately, how many times did you see any QoE Probe feedback form while you were using Flexisketch?

Q12: How good was the QoE Probe in trying in minimizing the disturbance of your modelling work? [Opinion Score Scale, rationale]

Q13: Any other comment?

PART 3

Gathering of Monitoring Data

Chapter 4 : KPIs in Software Ecosystem: A Systematic Mapping Study

[The chapter is based on: ***F. Fotrousi**, S. Fricker, M. Fiedler (2014). "KPIs in Software Ecosystem: A Systematic Mapping Study", 5th International Conference on the Software Business (ICSOB), Paphos, Cyprus: Springer, pp 194-211. DOI: 10.1007/978-3-319-08738-2.*]

Abstract

To create value with a software ecosystem (SECO), a platform owner has to ensure that the SECO is healthy and sustainable. Key Performance Indicators (KPI) are used to assess whether and how well such objectives are met and what the platform owner can do to improve. This paper gives an overview of existing research on KPI-based SECO assessment using a systematic mapping of research publications. The study identified 34 relevant publications for which KPI research and KPI practice were extracted and mapped. It describes the strengths and gaps of the research published so far, and describes what KPI are measured, analysed, and used for decision-making from the researcher's point of view. For the researcher, the maps thus capture state-of-knowledge and can be used to plan further research. For practitioners, the generated map points to studies that describe how to use KPI for managing of a SECO.

Keywords

Software ecosystem, digital ecosystem, performance indicator, KPI, success factor, systematic mapping

1. Introduction

A software ecosystem (SECO) is about “the interaction of a set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationship among them” (Jansen et al. 2009). We include here any ecosystem that is based on or enabled by software, including pure software, software-intensive systems, mobile applications, cloud, telecommunications, and digital software ecosystems. The inclusion of telecommunications, for example, is important as many modern software services can only be realized with appropriate ICT infrastructure. Companies adopt a SECO strategy to expand their organizational boundaries, to share their platforms and resources with third parties, and to define new business models (Manikas and Hansen 2013b; Weiblen et al. 2012).

A SECO is frequently supported by a technological platform or market that enables the SECO actors in exchanging information, resources, and artefacts. Ownership of such a platform gives strategic advantages over the other SECO actors. It allows satisfying ever-increasing customer demands with limited own resources. It also allows improving one’s own knowledge about the marketplace. Such knowledge is necessary for innovation, evolution of a product or service offering, and identification of revenue opportunities (Barbosa and Alves 2011; IBosch 2009).

SECO platform ownership also brings responsibilities. These include the definition of SECO performance objectives and management of the SECO to achieve these objectives. A SECO is expected to be healthy (Costanza and Mageau 1999) and sustainable (Chapin et al. 1996). It is healthy when it is productive for surrounding actors, robust, and niche-creating (Iansiti and Richards 2006). It is sustainable when it maintains its structure and functioning in a resilient manner (Costanza and Mageau 1999). Health and sustainability are closely linked performance objectives (Rapport et al. 1998) that are often found in complex systems (Costanza 1992).

Managing a SECO involves definition of how actors, software, and business models play together to achieve the SECO objectives

(Manikas and Hansen 2013a) in business, technical, and social dimensional perspectives (Santos et al. 2012). The platform owner uses performance indicators for benchmarking and monitoring the resulting ecosystem behavior. Key performance indicators (KPI) are those among the many possible indicators that are important, easily measurable quantitatively or with an approximation of qualitative phenomena (Parmenter 2010). The KPI serve as early warnings about potentially missed SECO objectives (Westin 1998) and to detect patterns that are useful for predicting health and sustainability of the SECO (Cokins 2009). Any deviation from success baselines are recorded and acted upon to ensure that the main ecosystem's objectives are met.

The here presented study gives an overview of literature on KPI for software ecosystems. A systematic mapping methodology was followed to identify and classify publications based on the reported research and based on KPI use. The dimensions used for classifying research were the type of ecosystem that was studied and the type of result that was delivered by the research. The dimensions used for classifying KPI use were the researched KPI types, the SECO objectives these KPI were used for.

The knowledge gap for collecting evidences about KPI studies motivated to systematically evaluate distribution of studies and provide guidance for future improvement. For practitioners, the generated map describes how to use KPI in the management of a SECO. It enables the platform owner in understanding the indicators that are important to assess for given SECO objectives. For researchers, the generated map describes state of research and helps finding research gaps for understanding the definition and use of SECO KPI.

The remainder of the paper is structured as follows. Section 2 presents the research objectives and defines research questions, search strategy, study selection, and study quality assessment. Sections 3 and 4 present the results by giving an overview of SECO KPI research, respectively SECO KPI practice. Section 5 discusses the results. Section 6 summarizes and concludes.

2. Research Methodology

The goal of this study is to provide an overview of the research performed to investigate the use of KPI for managing software ecosystems. The systematic mapping approach (Petersen et al. 2008) allows to map the frequencies of publications over categories to see the current state of research. It also exposes patterns or trends of what kind of research is done, respectively has been ignored so far. Mapping the research results, in addition to the type of research, reveals researchers' current understanding of KPI-related practice.

2.1. Research Questions

To provide an overview on publications relevant to KPI use for SECO, two sets of research questions are defined in Table 4-1. With the first set of questions we mapped foci and gaps of research about SECO KPI. With the second set we mapped the state of practice that was reported by the research.

2.2. Systematic Mapping Approach

To answer RQ1, RQ3, we followed the systematic mapping guidelines proposed by Petersen (Petersen et al. 2008). We (i) conducted database search with a search string that matched our research scope, (ii) performed screening to select the relevant papers, (iii) built a classification scheme based on keywording the papers' titles, abstracts, and keywords, and (iv) used this classification scheme to map the papers.

To answer RQ2, we modified the mapping process by using the pre-existing classification schemes already used in (Petersen et al. 2008; Wieringa et al. 2006). For RQ4, we built the classification scheme by extracting keywords from the main body of the papers and aligning the emerging scheme with the relevant software industry standard. The research steps are explained below.

Table 4-1. Research questions

SECO KPI Research	**Rationale**
RQ1: What kinds of ecosystems were studied?	The answer to this question shows the intensity of SECO KPI research across application domains and types of ecosystems. Skewedness, e.g. due to a focus on just a few types of application domains and ecosystems, indicates gaps where additional research is needed.
RQ2: What types of research were performed?	The answer to this question shows the maturity of SECO KPI research. The more disproportioned conceptual solutions and empirical validation research are, the more there is a need for research that compensates.
Ecosystem KPI Practice	**Rationale**
RQ3: What objectives were KPI used for?	The answer to this question shows the purposes of SECO KPI. It allows understanding when a SECO is considered to be successful and when not. Correlation with the answer to RQ4 allows understanding how the satisfaction of these SECO objectives is measured.
RQ4: What ecosystem entities and attributes did the KPI correspond to?	The answer to this question gives an overview of relevant KPI that are used to assess achievement of SECO objectives. The KPI show how SECO objectives are operationalized and quantified. Skewedness, a focus on just one or a few KPI, may indicate the degree of universality the KPI have for SECO management.

(i) Database search. The study defined the following search strategy.

Search String. To get an unbiased overview of KPI use in SECO, the search string was created with keywords that capture population only. The first aspect used to define the population was the ecosystems that can be found in a software context: software, digital, mobile, service, cloud, telecommunication, and ICT ecosystems. We also included papers that focused on software supply by adding software supply to the search string. The second aspect used to define the population was the application or use of KPI. We used the terms indicators, metrics, measurements, success factors, key characteristics, and quality attributes as synonyms for KPI. To avoid bias about RQ3, we did neither constrain for what purpose information was gathered and used. To build a broad overview of the research area and avoid bias, no keywords were defined in relation to intervention (e.g. monitoring), outcomes (e.g. improvements to a SECO), or study designs (e.g. case studies).

The search string was built by concatenating the two population aspects with the *AND* operator. The search string was formulated as follows: *software OR (software-intensive) OR digital OR mobile OR service OR cloud OR communic* OR telecom* OR ict) PRE/0 (ecosystem* OR "supply network*") AND (measur* OR kpi* OR metric* OR analytic* OR indicator* OR "success factor*" OR "quality attribute*" OR "key characteristic*".*

Search Strategy. The papers were identified using the important research databases in software engineering and computer science including Scopus, Inspec, and Compendex, which support IEEEXplore and ACM Digital Library as well. The search string was applied to title, author's keywords and abstract of these papers. The search did not restrict the date of the publication.

Validation. We validated the set of identified papers by checking it against the papers used in the SECO literature reviews performed by (Barbosa and Alves 2011; Manikas and Hansen 2013b). Each paper used by these studies that was relevant for our study had been found by following the above-outlined database search.

(ii) Screening of papers. The inputs for this step were the set of papers identified with step (i). The first and second authors screened these papers independently We screened these papers to exclude studies that do not relate to the use of KPI for any ecosystem-related purpose and to ensure broad-enough coverage of the population. We describe here a complete set of inclusion and exclusion criteria.

Inclusion. We included peer-reviewed journal, conference, or workshop papers that were accessible with full text. The included papers describe the use of KPI in an ecosystem context or the effects of such KPI on properties of the ecosystem. Due to the importance of networking infrastructure and digital information exchange for a well-functioning software ecosystem we included telecommunication and information technology papers in addition to pure SECO papers.

Exclusion. We excluded papers that focused on the use of KPI for managing a member of the ecosystem only. For example, papers about the use of indicators for managing a single company that participates in the ecosystem, or a product or process of that company, were excluded because of their too narrow focus. We excluded papers that focused on other ecosystems rather than a software ecosystem. For example papers focus on biology, environmental, climate, and chemical aspects were excluded. When the definition of software ecosystem did not fulfill in the papers, they were excluded. As an example, the paper that considered Bugzilla and email system as software ecosystems was excluded, since such systems do not address the shared market concept of a SECO definition. Papers that study qualitative indicators using qualitative approaches such as a structured interview were excluded. Also, we excluded papers that focused on ecosystem design in place of ecosystem management. For example, papers about the design of interoperability protocols or of products or services offered to an ecosystem were excluded. The papers that do not Finally, to avoid inclusion of papers that only speculated about KPI use or effects, we excluded papers that did not report any empirically-grounded proof-of-concept.

(iii) Building the classification scheme. To answer the research questions RQ1, RQ3, and RQ4 we employed keywording (Petersen et al. 2008) as a technique to build the classification scheme in a bottom-up manner. Extracted Keywords were grouped under higher categories to make categories more informative and to reduce number of similar categories. We built the ecosystem classification scheme by extracting the types and application domains of the studied ecosystems. We built the classification scheme for KPI practice by extracting KPI assessment objectives, entities and attributes used for measuring the KPI.

The keywords were extracted from the papers' titles, keywords, and abstracts. When the quality of an abstract was too poor, we used the main body of the paper to identify the keywords. Similarly, as most of the papers did not included sufficient information about entities and attributes measured with KPI inside the abstract, we used the main body of the papers for keyword identification. The keywords obtained from extraction were then combined and clustered to build the categories used for mapping the papers. The clustering of measurement attributes was aligned with the categories described in ISO/IEC FDIS 25010 as far as applicable.

To answer RQ2, we used a pre-defined classification scheme (Wieringa et al. 2006) that was used by earlier systematic mapping studies (Petersen et al. 2008). It classifies research types into validation research, evaluation research, solution proposals, philosophical papers, opinion papers, and experience papers.

(iv) Systematic mapping of the papers. When the classification scheme was in place, the selected papers were sorted into the classification scheme. The classifications were then calculated the frequencies of publications for each category.

To answer RQ1 and RQ2 we reported the frequencies of the selected papers for the categories in the dimensions of ecosystems types and application domains, respectively in the dimensions of research type and research contributes type. We used x-y scatterplots with bubbles in category intersections to visualize the kinds of ecosystems that were studied. The size of a bubble is depicted proportional to the number of papers that are in the pair of categories that correspond to the bubble coordinates. The visualized frequencies make it possible to see which categories have been emphasized in past research and which categories received little or no attention.

To answer RQ3, we first described the categories identified when building the classification scheme and how these categories were expressed in the selected papers. This description resulted in a dictionary for interpreting the scatterplots used for describing how

SECO KPI are used in relation to these objectives. We again used x-y scatterplots for showing the frequency of pairs of categories. These pairs allowed us to describe the attributes measured for each type of ecosystem entity, the measurements used in relation to the SECO objectives, and how KPI are obtained for various kinds of entities found in a SECO.

2.3. Threats to Validity

This section analyses the threats to validity for the taxonomies of construct, reliability, internal and external validity.

Construct validity reflects whether the papers included in the study reflect the SECO KPI phenomenon that was intended to be researched. The search string was constructed in an inclusive manner so that it captured the wide variety of software-related ecosystems and the many different names given to key performance indicators. The common databases, used for software and management-related literature research, were used to find papers. Only after this inclusive process, manual screening was performed to exclude papers not related to the research objectives. The list of included papers was then validated against two systematic studies on software ecosystem (Barbosa and Alves 2011; Manikas and Hansen 2013b) and found that the review covers all relevant papers.

Reliability validity refers to the repeatability of the study for other researchers. The study applied a defined search string, used deterministic databases, and followed a step-by-step procedure that can be easily replicated. The stated inclusion and exclusion criteria were systematically applied. Reliability of the classification was achieved by seeking consensus among multiple researchers.

Internal validity treats refers to problems in the analysis of the data. These threats are small, since only descriptive statistics were used.

External validity concerns the ability to generalize from this study. Generalization is not an aim of a systematic mapping study as only one state of research is analyzed and the relevant body of research completely covered. In particular, the study results about the use of SECO KPI, reflects the practices studied in SECO KPI research and not SECO KPI practice performed in general.

3. Results: Ecosystem KPI Research

The database search resulted in a total of 262 papers, including 46 duplicates. After screening and exclusion, 34 papers remained and were included in the study. These selected papers were published from 2004 onwards. This section gives an overview of the research described in the selected papers. Appendix A lists the selected papers.

3.1. Kinds of Ecosystems

To answer RQ1, Figure 4-1 gives an overview over the ecosystems that our study found KPI research for. The number embedded in a bubble indicates how many papers were devoted to a given combination of ecosystem type and application domain (multiple classifications possible). Empty cells indicate that no corresponding study was found. The number on the category label indicates the total number of papers in that category.

Most of the papers used the term software ecosystem to characterize the studied ecosystems. Special kinds of ecosystems were cloud, service, mobile apps, and open source software ecosystems. Less frequent were digital ecosystems with 44% of the papers. They refer to the use of IT to enable collaboration and knowledge exchange (Boley and Chang 2007).

The papers addressed a variety of application domains. Most common were telecommunications, business management and software development. None of the remaining application domains was addressed by more than one or two papers. Thus, research is rather scattered, and the specifics of the various application domains only little understood.

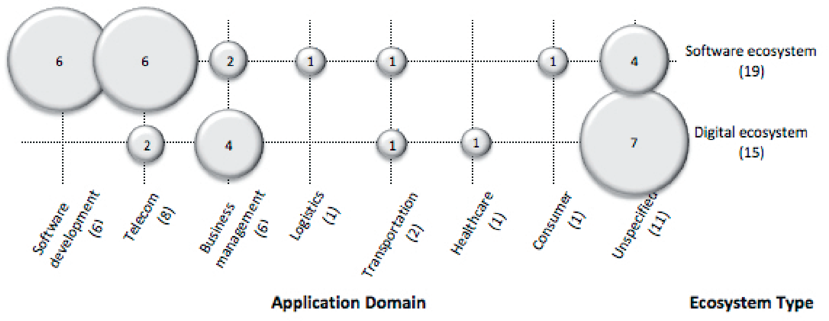


Figure 4-1. Kinds of ecosystems that were studied with KPI research. The label "software ecosystem" refers to those that are not considered a digital ecosystem (see main text).

3.2. Types of Research

To answer RQ2, Figure 4-2 presents a map of the kind of research performed on KPI in software-related ecosystems. Papers with multiple research types and contributions were classified for each combination of research type and contribution they presented.

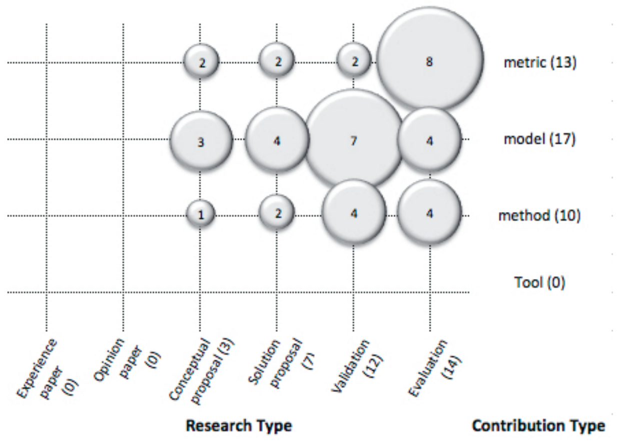


Figure 4-2. Map of research on SECO KPI and type of contributions.

Experience report papers describe experiences in working with SECO KPI and usually describe unsolved problems. *Opinion papers* discuss opinions of the papers' authors. *Conceptual proposal* papers sketch new conceptual perspectives related to SECO KPI. This category renamed *philosophical papers* category (described in iii of section 2.2) to fit the SECO KPI study. *Solution proposal* papers propose new techniques or improve existing techniques using a small example or a good argumentation. *Validation* papers investigate novel solutions that had not been implemented in practice (e.g. experiment, lab working). *Evaluation* papers report on empirical or formal studies performed to implement a solution or evaluate the implementation.

Metric papers describe KPI for SECO. *Model* papers describe relationships between KPI. *Method* papers describe approaches for working with SECO KPI. Finally, *tool* papers describe support for work with SECO KPI.

Most research was found in the categories of validation and evaluation. Research contributed with metrics, models, or methods. For example, R17 proposes a model that explains how health can be measured with relevant indicators (conceptual proposal, model) and validates that model with a questionnaire (validation, model). R14 proposes a method for assessing services based on Quality of Service indicators (solution, method). R19 evaluates factors that affect successful selling in e-markets (metric, evaluation). No paper was an experience report or an opinion paper. No paper contributed with any tool.

4. Results: Researched KPI Practice

The papers included in this study describe the use of KPI by a platform owner for achieving objectives with the ecosystem that was enabled by the ecosystem platform. This section gives an overview of these objectives and the KPI that were used.

4.1. Ecosystem Objectives Supported by KPI

KPI were used to enable or achieve a variety of objectives. Platform owners aimed, at improving business, at improving the interconnectedness between actors, at growing the ecosystem, at improving quality of ecosystem, product, or services performed within the ecosystem, and at enabling sustainability of the ecosystem (answer RQ3):

Business improvement. Research has been performed on how to improve business at the ecosystem level. The studied business improvements concerned the perspectives of ecosystem activity and of commercial success. Ecosystem activity related to the level of activity of participating actors, encouragement to participate in the ecosystem, and the transaction volume. Commercial success related to sales success, innovativeness and competitiveness of the participating actors, and the cost of the network that enables the ecosystem. The activity and commercial perspectives were mixed in the papers, thus could not be separated in the analysis of the literature.

Interconnectedness improvement. Research has been performed on how to improve interaction in an ecosystem, for example to reduce cost, improve predictability of services that are provided in the ecosystem, and manage trust. Interaction improvement was studied between individual actors and between whole networks contained in the ecosystem. The research differed in terms of lifecycle stage of an interaction and covered supplier availability, discovery, ranking and selection, the resulting connectivity, interaction evaluation, and the impact of the interaction on the actors that participated in it. Interaction improvement was not always an end in itself but was considered essential for generating business activity and sustainability of the ecosystem.

Growth and stability. Research has been performed on how to manage growth and stability of the ecosystem. Growth and stability were seen as two factors that need to be managed jointly. During growth flexibility and controllability need to be maintained. During stability, a continuous co-revolution must happen. Growth and stability again are not ends in themselves, but thus contribute to sustainability and survival of the ecosystem.

Quality improvement. Research has been performed on how to manage quality of ecosystems. In particular, performance, usability, security, data reliability, extendibility, transparence, trustworthiness, and quality-in-use were investigated. Quality management was sometimes presented as an ends in itself, for example by allowing comparison among multiple ecosystems, enabling diagnosis, improving decision-making, and achieving long-term usage of services. At the same time, however, quality management was considered to be a means to encourage adoption and growth, improve business performance, and achieve sustainability.

Enable sustainability. Research has been performed on how to sustain an ecosystem. Two angles were taken: self-organization and resource consumption. Self-organization was approached through continuous rejuvenation of the ecosystem. Resource consumption was studied in relation of electrical energy. Throughout all papers found in this category, sustainability was considered to be desirable ends for software ecosystems.

4.2. KPI: Measured Entities

The included papers describe measurements applied to the ecosystem as a whole or the parts the ecosystem consists of: actor, artifact, service, relationship, transaction and network.

Actors. Actors were measured and characterized as follows. They were human or artificial. Examples of human or legal actors were sellers and developers that provide products to buyers or groups of organizations and firms. Examples of artificial actors were nodes in a telecommunication network. An actor engages in transactions in an ecosystem and builds relationships to other actors or artifacts. The transactions the seller engages in generate profit and revenue for the cost the seller is willing to take. Effective actors have knowledge about other actors or the network and has good interestingness and reputation for other actors. Actors are also considered to be sources and sinks of data and have differing ranges for data transmission. Performance of individuals and groups in terms of fulfilled tasks and decisions as well as performance of firms and organizations in terms of profits are measured.

Artifacts. Artifacts such as software, codes, plugins, books, music, or data were measured and characterized as follows. Artifacts had a location in the ecosystem. They evolve, may have reputation and popularity, and exposed their consumers to vulnerability.

Services. Services were measured and characterized as follows. Services consume energy and other resources. Services have quality attributes such as quality of service, security, compliance, and reputation. Metadata and service level agreements are used to specify the services. The services are not fixed but evolve: services emerge, change, and get extinct. A special service was provided by the platform that laid the fundament for the ecosystem. It was characterized in terms of attributes like stability, documentation, portability, and openness.

Relationship. Relationships were measured and characterized as follows. Actors enter relationships with other actors, artifacts, or services. A relationship connects two or more such entities. Examples of relationships were business connections and telecommunication communication links. A relationship may be transparent and express a trust value of the connected entities. A relationship is the basis for transactions, thus is used for advertising and building alliances. The transaction, however, is constrained by cost and quality of the relationship.

Transactions. Transactions were measured and characterized as follows. Examples of transactions are sales of services to customers, server requests, and commits of code files made by developers. They are initiated with an offer that is measured in terms of attributes like price and quantity. Transactions also have a price and quantity. Other attributes include time to negotiate the transaction, time to complete, energy consumption, transmission rate, and buyer satisfaction.

Network. Networks were considered as sets of entities and relationships that were part of a whole ecosystem. Examples were local or application-specific networks. Networks were characterized as follows. Networks were vulnerable to security threats such as data availability, integrity, authentication, and authorization. Networks differed in the node density, degree of collaboration, provisioning cost, and hit rate for artifacts.

Ecosystem. Full ecosystems were characterized as follows. They have quality attributes like size, performance, security and energy consumption that can also characterize networks contained in an ecosystem. In addition, ecosystems exhibited lifelines, diversity, stability, transparency, healthiness, and sustainability.

This section and next section collaboratively provide answer for RQ4. The map in the left part of Figure 4-3 shows the entities that were studied in relation to the ecosystem objectives. Most research studied the measurement of the overall ecosystem to enable quality or business improvement. For example, R17 describes how performance of the ecosystem affected user satisfaction, and R13 shows how analytics applied to the ecosystem can be used to improve business. Considerable research was also devoted to improving the interconnectedness of the ecosystem, where attributes of the products and services played an important role and also to the role of platform measurements to grow the ecosystem and improve quality. For example, R6 described how to use a service similarity measurement was used to improve ecosystem connectivity. R2 described how growth, diversity, and entropy measurements of a SOA platform were used to increase growth. R4 described how communication quality measurements were used to improve the quality of a telecommunication ecosystem.

The map also shows areas where no research was published. For example, no research studied the role of network measurements for objectives other than sustainability and quality improvement.

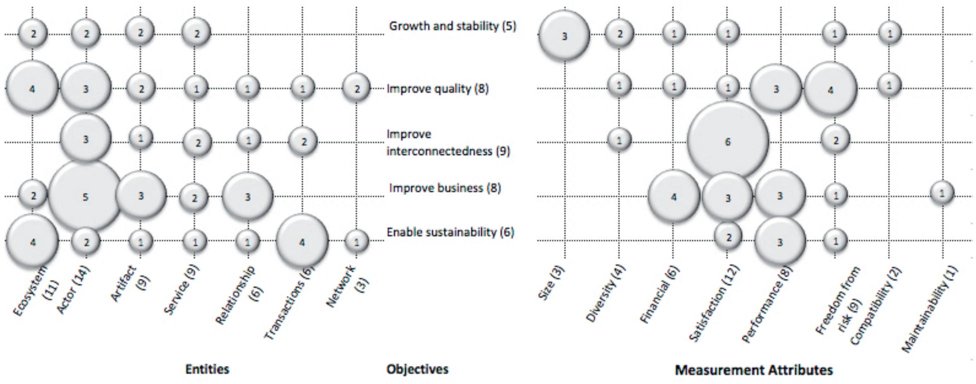


Figure 4-3. Map of measured entities and measurement attributes in relation to ecosystem objectives.

4.3. KPI: Measurement Attributes

To make the state and evolution of the ecosystem and of its elements visible, a broad variety of attributes were measured.

The following attributes categories emerged when clustering the attributes described in the included papers. Figure 4-4 shows how classes of quality attributes were merged toward new categories. The *size* category includes attributes to measure size and growth. *Diversity* includes attributes to measure heterogeneity and openness for such heterogeneity. *Financial* includes attributes to measure economic aspects such as investment, cost, and price. *Satisfaction* includes attributes to measure satisfaction and the related concepts of suitability, interestingness, learnability, usability, accessibility, acceptability, trust, and reputation. *Performance* includes attributes to measure performance, including resource utilization, efficiency, accuracy, and effectiveness. *Freedom from risk* includes attributes to measure the ability to avoid or mitigate risks and includes the related concerns of security, reliability, maturity, availability, and other related guarantees. *Compatibility* includes attributes to measure the degree to which an entity can perform well in a given context, interoperate or exchange information with other entities, and be ported from one context to another one. *Maintainability* includes attributes to measure flexibility, respectively the ability to be changed.

The right part of Figure 4-3 gives an overview of the attributes referred to by KPI. Most research studied measurements of satisfaction, typically to improve business or interconnectedness. An example of such research is R13 that describes the use of seller reputation to improve business. To support quality improvement, all measurement attributes that relate to quality were included in at least one research paper, except for maintainability and size. Similarly, size measurements did not play any role other than for growth and stability.

- Diversity
  - Heterogeneity
  - Openness
- Satisfaction
  - Satisfaction
  - Suitability
  - Interestingness
  - Learnability
  - Usability
  - Accessibility
  - Acceptability
  - Trust
  - Reputation.
- Performance
  - Performance
  - Resource utilization
  - Efficiency
  - Accuracy
  - Effectiveness
- Financial
  - Investment
  - Cost
  - Price
- Size
  - Size
  - Growth
- Freedom from risk
  - Risk mitigation
  - Security
  - Reliability
  - Maturity
  - Availability
  - Guarantees.
- Compatibility
  - Interoperability
  - Exchangeability
- Maintainability
  - Flexibility
  - Changeability

Figure 4-4. Merging classifications of measurement attributes

The left part of Figure 4-5 shows how the ecosystem elements were measured. Satisfaction was a common attribute that was measured for any entity except for rules. This shows that a same attribute can be measured or analysed for different ecosystem entities. Also, it is revealed that similar measurement attributes might be collaborating to measure different ecosystem elements. As an example, CCCI (correlation, commitment, clarity and importance) measurable attributes were used to measure trust as well as reliability.

The overall ecosystem and actors were the most comprehensively measured or analyzed entities, with a special focus on satisfaction, freedom from risks and performance. Some examples of such satisfaction measurements are provided by R13 that measured usage and acceptability of an ecosystem. The service followed with the next largest variety of measurements. R2, for example, measured entropy and diversity to characterize platform complexity. Only narrow sets of measurement attributes were applied to the business partner, interactions, and business.

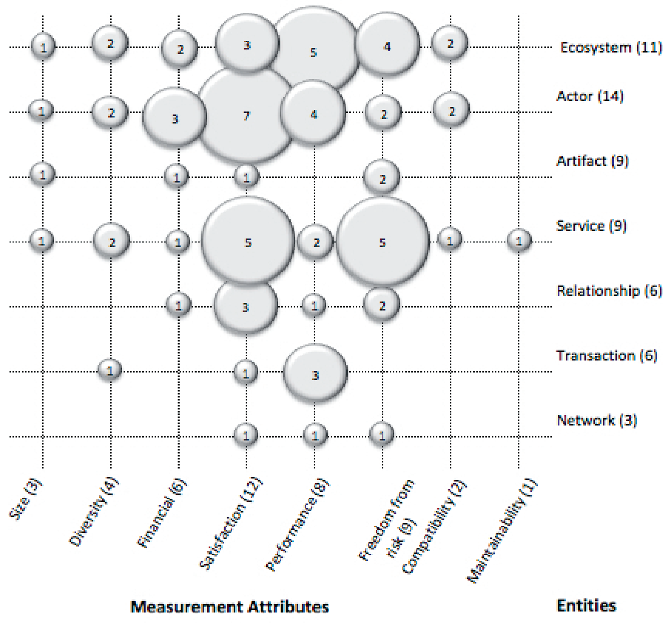


Figure 4-5. Map of measurement attributes in relation to the measured entities.

5. Discussion

The study provides a classification of KPI relevant papers in understanding researches, relationship with the practice, and assessment of research outcomes. This classification contributes to taxonomy, which can help for closer examination of the ecosystem or platform owner objectives, making them more recognizable in designing KPI. New KPI can be extracted for an ecosystem using this taxonomy, and existing KPIs can be extended or restructured applying the generic structure of the taxonomy.

The literature map indicates that KPI for software-based ecosystems is a thin area with work at all maturity levels. Journal, conference, and workshop papers exist. However, the number of publications is not sufficient, and many application domains for ecosystems addressed with just one or two papers. Although formulation of KPI might be domain dependent and similarity of objectives is not the only factor to select a KPI, however due to insufficient study it is difficult to state whether characteristics of a domain, for example regulation of healthcare, affects the KPI of the ecosystem that targets that domain.

The included research on ecosystem KPI mostly addresses ecosystem measurements or measurements of satisfaction, performance and freedom from risks. Measurements other than satisfaction that are applied on elements contained in the ecosystem are comparatively little researched. A broader understanding of KPI would increase a platform owner's flexibility in measuring, analyzing, and using KPI for decision-support. The understanding of a greater variety of KPI would also contribute to increased transparency of status, evolution, and other aspects of the ecosystem.

6. Conclusion

The here presented study gives an overview of literature on the use of KPI for software-based ecosystems. A systematic mapping methodology was followed and applied to 34 included studies published from 2004 onwards.

To respond to RQ1 and RQ2, research was broad but thin. Two major kinds of ecosystems were researched: software ecosystems and digital ecosystems. Many application domains were addressed, but most of them with one or two papers only. The published research was mature with journal, conference, and workshop papers that covered metrics, models, and methods. In response to RQ3 and RQ4, KPI research was skewed. Most research studied ecosystem KPI for improving the interconnectedness between individual actors and subsystems of the ecosystem. Overall, most KPI were about satisfaction, performance and freedom from risks measures.

The results of the mapping study indicate that more research is needed to better understanding of KPI for software-based ecosystems. In particular, a deeper understanding of how the application domain affects an ecosystem's KPI is needed. Also, an important research opportunity is the identification, analysis, and evaluation of KPI. Such research could make the work with KPI more flexible, because a greater variety of KPI would be known and available for the practitioner to use.

Appendix

The Selected Studies of the systematic mapping researc.

ID	References
R1	Sabry, N., Krause, P.: A digital ecosystem view on cloud computing. 6th IEEE International Conference on Digital Ecosystems Technologies (DEST). Piscataway, NJ, USA (2012)
R2	Fiegler, A., Dumke, R.R.: Growth-and Entropy-Based SOA Measurement: Vision and Approach in a Large Scale Environment. Software Measurement,Joint Conference of the 21st Int'l Workshop on and 6th Int'l Conference on Software Process and Product Measurement (IWSM-MENSURA). Los Alamitos, CA, USA (2011)
R3	Pranata, I., Skinner, G., Athauda, R.: TIDE: Measuring and evaluating trustworthiness and credibility of enterprises in digital ecosystem. International Conference on Management of Emergent Digital EcoSystems. San-Francisco, USA (2011)
R4	Yang, Y., Xu, Y., Li, X., Chen, C.: A loss inference algorithm for wireless sensor networks to improve data reliability of digital ecosystems. Industrial Electronics, IEEE Transactions on 58, 2126-2137 (2011)
R5	Savola, R.M., Sihvonen, M.: Metrics driven security management framework for e-health digital ecosystem focusing on chronic diseases. International Conference on Management of Emergent Digital EcoSystems. Addis Ababa, Ethiopia (2012)
R6	Dong, H., Hussain, F.K., Chang, E.: A service concept recommendation system for enhancing the dependability of semantic service matchmakers in the service ecosystem environment. Journal of Network and Computer Applications 34, 619-631 (2011)
R7	Barolli, L., Yang, T., Mino, G., Durresi, A., Xhafa, F.: A simulation system for WSNs as a Digital Eco-System approach considering goodput metric. 4th IEEE International Conference on Digital Ecosystems and Technologies (DEST). Dubai, United Arab Emirates (2010)
R8	Nankani, E., Simoff, S., Denize, S., Young, L.: Enterprise university as a digital ecosystem: Visual analysis of academic collaboration. 3rd IEEE International Conference on Digital Ecosystems and Technologies, DEST'09. Istanbul, Turkey (2009)
R9	Fabregues, A., Madrenas-Ciurana, J., Sierra, C., Debenham, J.: Supplier performance in a digital ecosystem. 3rd IEEE International Conference on Digital Ecosystems and Technologies, DEST'09. Istanbul, Turkey (2009)
R10	van den Berk, I., Jansen, S., Luinenburg, L.: Software ecosystems: a software ecosystem strategy assessment model. Fourth European Conference on Software Architecture. ACM, Copenhagen, Denmark (2010)

ID	References
R11	Taghizadeh, M., Plummer, A., Aqel, A., Biswas, S.: Towards optimal cooperative caching in social wireless networks. Global Telecommunications Conference (GLOBECOM). IEEE, Miami, Florida, USA (2010)
R12	Dong, H., Hussain, F.K., Chang, E.: Semantic service retrieval and QoS measurement in the digital ecosystem environment. International Conference on Complex, Intelligent and Software Intensive Systems (CISIS). Krakow, Poland (2010)
R13	Tian, C.H., Cao, R.Z., Zhang, H., Li, F., Ding, W., Ray, B.: Service analytics framework for web-delivered services. International Journal of Services Operations and Informatics. 4, 317--332 (2009)
R14	Chen, W., Chang, E.: A method for service quality assessment in a service ecosystem. International Conference on Digital Ecosystems and Technologies Inaugural IEEE. Piscataway, NJ, USA (2007)
R15	Koendjbiharie, S., Koppius, O., Vervest, P., van Heck, E.: Network transparency and the performance of dynamic business networks. 4th IEEE International Conference on Digital Ecosystems and Technologies (DEST). Dubai, United Arab Emirates (2010)
R16	Jansen, S.: How quality attributes of software platform architectures influence software ecosystems. International Workshop on Ecosystem Architectures. Saint Petersburg, Russian Federation (2013)
R17	Salem, A.M.B.H., Ghadhab, B.B.: Performance Measurement practices in Software Ecosystem. International Journal of Productivity and Performance Management. 62, 514 - 533 (2013)
R18	Goeminne, M., Mens, T.: A framework for analysing and visualising open source software ecosystems. Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE). Antwerp, Belgium (2010)
R19	Pereira, A., Duarte, D., Meira Jr, W., Góes, P.: Assessing success factors of selling practices in electronic marketplaces. International Conference on Management of Emergent Digital EcoSystems. Lyon, France (2009)
R20	Dong, H., Hussain, F.K., Chang, E.: A QoS-based service retrieval methodology for digital ecosystems. International Journal of Web and Grid Services 5, 261-283 (2009)
R21	Fachrunnisa, O., Hussain, F.K.: A methodology for maintaining trust in industrial digital ecosystems. IEEE Transactions on Industrial Electronics 60, 1042-1058 (2013)
R22	La, H.J., Kim, S.D.: A model of quality-in-use for service-based mobile ecosystem. 1st International Workshop on the Engineering of Mobile-Enabled Systems (MOBS). IEEE, San Francisco, CA, USA (2013)
R23	Ion, M., Danzi, A., Koshutanski, H., Telesca, L.: A peer-to-peer multidimensional trust model for digital ecosystems. 2nd IEEE

ID	References
	International Conference on Digital Ecosystems and Technologies (DEST). IEEE, Phitsanuloke, Thailand (2008)
R24	Enokido, T., Aikebaier, A., Takizawa, M.: An integrated power consumption model for communication and transaction based applications. International Conference on Advanced Information Networking and Applications (AINA). Biopolis, Singapore. IEEE (2011)
R25	Wright, J.L., McQueen, M., Wellman, L.: Analyses of two end-user software vulnerability exposure metrics (extended version). Information Security Technical Report 17, 173-184 (2013)
R26	Böhmer, M., Ganev, L., Krüger, A.: Appfunnel: A framework for usage-centric evaluation of recommender systems that suggest mobile applications. International conference on Intelligent user interfaces. ACM, Santa Monica, CA, USA (2013)
R27	Eklund, U., Bosch, J.: Architecture for embedded open software ecosystems. Journal of Systems and Software - Article in Press (2014)
R28	Zhang, J., Liang, X.J.: Business ecosystem strategies of mobile network operators in the 3G era: The case of China Mobile. Telecommunications Policy 35, 156-171 (2011)
R29	Walden, J., Doyle, M., Lenhof, R., Murray, J., Plunkett, A.: Impact of plugins on the security of web applications. 6th International Workshop on Security Measurements and Metrics. ACM, Bolzano-Bozen, Italy (2010)
R30	Straub, D., Rai, A., Klein, R.: Measuring firm performance at the network level: A nomology of the business impact of digital supply networks. Journal of Management Information Systems 21, 83-114 (2004)
R31	Vasilescu, B., Serebrenik, A., Goeminne, M., Mens, T.: On the variation and specialisation of workload-A case study of the Gnome ecosystem community. Empirical Software Engineering - Article in Press (2013)
R32	Luna, J., Ghani, H., Vateva, T., Suri, N.: Quantitative Assessment of Cloud Security Level Agreements: A Case Study. 7th International Conference on Security and Cryptography. SECRYPT. INSTICC Press, Setubal, Portugal (2012)
R33	van Angeren, J., Blijleven, V., Jansen, S.: Relationship intimacy in software ecosystems: a survey of the dutch software industry. International Conference on Management of Emergent Digital EcoSystems. ACM, San Francisco, CA, USA (2011)
R34	Liu, Y., Fan, Y., Huang, K.: Service Ecosystem Evolution and Controlling: A Research Framework for the Effects of Dynamic Services. International Conference on Service Sciences (ICSS). IEEE, Shenzhen, China (2013)

Chapter 5 : Software Analytics for Planning Product Evolution

[The chapter is based on: *F. Fotrousi, S. Fricker (2016). "Software Analytics for Planning Product Evolution", 7th International Conference of Software Business (ICSOB), Ljubljana, Slovenia: Springer, pp. 16-31. DOI: 10.1007/978-3-319-40515-5_2.*]

Abstract

Evolution of a software product is inevitable as product context changes and the product gradually becomes less useful if it is not adapted. Planning is a basis to evolve a software product. The product manager, who carries responsibilities of planning, requires but does not always have access to high-quality information for making the best possible planning decisions. The current study aims to understand whether and when analytics are valuable for product planning and how they can be interpreted to a software product plan. The study was designed with an interview-based survey methodology approach through 17 in-depth semi-structured interviews with product managers. Based on results from qualitative analysis of the interviews, we defined an analytics-based model. The model shows that analytics have potentials to support the interpretation of product goals while is constrained by both product characteristics and product goals. The model implies how to use analytics for a good support of product planning evolution.

Keywords

Analytics, Measurements, Product Planning, Software Evolution

1. Introduction

Software products are evolved throughout their life cycle through extension and adaptation of functionality and quality (Rajlich and Bennett 2000). Such evolution is inevitable as product context changes and a software gradually becomes less useful if it is not adapted (Lehman 1980). The flexibility of service-oriented approaches enables such evolution thinking (Gold et al. 2004). Early release of a minimal viable product followed by evolution is beneficial for the product organization because it allows increasing return on investment when compared with a late release of a near-perfect product (Choudhary 2007; Denne and Cleland-Huang 2004). Also, early release of a product allows learning about actual customer wants and needs; and the use of such market information in later product evolution is determinant for product success (Ottum and Moore 1997).

Mature companies plan how they intend to achieve their strategic objectives and satisfy market needs (Fricker et al. 2012; Kittlaus and Clough 2009). Planning concerns the product portfolio, the long-term roadmap of each product, and the short-term release plans (Bekkers et al. 2010). Portfolio management is about the strategic choice of which markets, products, and technologies the product organization addresses and, consequently, how it intends to spend its scarce resources on marketing, engineering, and research (Cooper et al. 1999). Roadmapping supports strategic and long-range planning for exploring evolving markets, products, and technologies and for coordinating the actions of the product organization to address opportunities and threats (Phaal et al. 2004). Release planning, finally, addresses the short-term time horizon by selecting an optimum set of features to be delivered in a release so that competing stakeholder demands, benefits for the product organization, and available resources are balanced (Svahnberg et al. 2010). The impact of product planning, in comparison to the absence of such planning, are shorter projects, fewer delays, and improved quality (Ebert 2007).

Product plans are based on information about company goals, market trends, product requirements, and stakeholder priorities

(Bekkers et al. 2010). That information is collected and the resulting plans validated by consulting company-external stakeholders such as customers, partners, and consultants that monitor the market and company-internal stakeholders such as marketing, sales, research, development, support, sales, and company board representatives. Many techniques exist for such consultation of stakeholders, including workshops (Phaal et al. 2007), focus groups (Krueger 2009), and surveys (Fowler 2009). Stakeholder consultation is essential for achieving clarity, support, and stability of the product vision and the plans that refine it (Lynna and Akgünb 2001).

Even-though stakeholder consultation is widely established and considered good practice; the value of information obtained by this approach is limited, especially in a context with many users and customers. The consulted representatives are intermediaries to the real stakeholders. Non-probabilistic sampling, especially convenience sampling, tends to produce biased input (Robson 2002). Even if a representative set of stakeholders is identified, it is questionable whether their expressed opinion corresponds to the actual interest. An expressed customer wish does not necessarily translate to a buying decision (Howard and Sheth 1969). Finally, dependency on stakeholders exposes the product manager to power and politics. Stakeholders exert their power by telling the product manager what to do and by creating a reality in which the product manager has to act according to these instructions (Milne and Maiden 2012). The resulting political decisions risk benefiting the most powerful of these stakeholders, but not necessarily the product.

This paper proposes the use of software analytics (Zhang et al. 2011) as a new source of information for product planning evolution. Analytics are the quantitative measures of an entity (Davenport and Harris 2007), which provide insight and actionable information (Zhang et al. 2011) for a data-driven decision making (Buse and Zimmermann 2010; Buse and Zimmermann 2012). Analytics have the potential to become useful decision-support for software made available to customers and users, but still is undergoing evolution. In contrast to stakeholder consultation, measurement of product use and quality provides evidence that is representative, unbiased, and free from power and politics.

Based on a review of existing literature on software product planning and analytics, the paper introduces a conceptual model that connects measurements of the software product to product planning decisions. The study explores the connection by discussing it in interviews with 17 software product managers. The Inductive content analysis method (Elo and Kyngäs 2008) was used to identify how the measurements would be interpreted and used for product planning decision-support. The results provide insights for method and tool engineering (Brinkkemper 1996) and for research targeted at simplifying product planning and improving the reliability product planning decisions.

This paper extends an earlier paper that presented the statistical analysis performed to understand product manager preferences for analytics (Fotrousi et al. 2013). The present paper gives an in-depth analysis whether and when analytics are valuable for product planning and how the interviewed product managers would use analytics for obtaining product planning decision-support for evolution.

The remainder of the paper is structured as follows. Section 2 reviews existing work in software analytics and introduces a conceptual model that describes how software analytics provide decision-support for product planning. Section 3 describes the research design used in the study. Sections 4 presents the empirical results and analyze the collected data. Section 5 discusses the results and their implications on practice and research. Section 6 summarizes and concludes the paper.

2. Background

Software product analytics are the quantitative measures, collected during product use, giving actionable insight (Zhang et al. 2011) for deciding about product evolution (Buse and Zimmermann 2010). The actionable insight characteristics of analytics differentiate it from measures or metrics terms, which are used interchangeably in literature (e.g. ISO-9126 used the term *metrics* but replaced by *measures* in ISO 15393). Some literature refers to analytics as the process of developing actionable insight (Cooper 2012). However, our definition emphasizes analytics as quantitative measures.

In product planning context, analytics measures a product, feature, or quality attribute. A product consists of features (Gorchels 2000) and each feature is composed of a set of functional and non-functional requirements (Fricker and Schumacher 2012). A product manager should deal with decision-making about creation, change, deletion, prioritization or allocation concerning product, features or requirements. Table 5-1 gives an overview of decisions that can be made during the planning of a software product. The decisions are distributed based on the practice areas including portfolio management, roadmapping, and release planning.

Table 5-1. Taxonomy of product planning decisions

Practice Area	**Decision Object**	**Decision Alternatives**						
Portfolio Management	Products in the company's portfolio	Create	Enhance, Change	Prioritize	Remove	Allocate Resource	Allocate to Release	Confirm Technology
Product Roadmapping	Features of a product							
Release Planning	Requirement in a feature selected for release							

The decisions of product planning have a strong relationship with software product delivery. The trend of changing the software delivery from packaged product to SaaS (Software as a Service) delivery model (Cusumano 2008) implies faster and smaller release of new features (Choudhary 2007), ease of developing more features upon request (Choudhary 2007) in addition to facilitating data collection to support planning decisions. SaaS delivery model enables monitoring of software use and provides first-hand information about market, attractiveness of software and its features.

Table 5-2. Taxonomy of measurements for SaaS-based applications

Mapped Entities to product	**Entities**	**Attributes**		
		Health	**Usage**	**Context**
Product	Website	Errors, Downtime, Response time, Throughput, Attacks	Use, Time between uses, Duration of use.	Users, New users, Returning users, Referrers, Location/ISP per use, Search engines and keywords, Campaigns, Browsers, Operating systems, Languages, Plugins, Screen resolutions.
Feature/ content	Page	Errors, Response time	Use, Time between uses, Duration of use, Entrance, Click activity, Depth of use, Click stream/path, Exit, Bounce.	Users, Search engines and keywords, Campaigns
GUI Requirement	GUI Element	-	Use, Time between uses, Click activity, Click stream/path.	-

Table 5-2 illustrates a taxonomy of the measurement attributes in SasS based products. For such products, the measurement attributes belong to entities such as a product, feature/content or GUI requirement that can be mapped to entities of a website, page or GUI element in a general web application. Product managers conceptualize a web application as a product that consists of features instead of pages. Page is the definable unit of content. A feature can be one page, part of a page or distributed among pages. A request for the feature can be defined as a page request. Similar to a feature, a page can be conceptualized as a content, since it provides an additional information resource for the feature contributing to the end user knowledge. In a SaaS-based product, functional requirements may belong to graphical elements of a feature (i.e. page) measurable for a GUI requirement entity.

The second part of the taxonomy presented in Table 5-2 categorizes the corresponding measurement attributes based on the measurement purpose for products' *health*, *usage* and *context*. The attributes corresponding to *health* of entities inform technical quality of services (Menasce 2002). The category of *usage* measurement attributes specifies the key data for understanding a traffic behavior of users (Srivastava et al. 2000) from the entity-use perspective. Context measurement attributes address the circumstances of users or sources in which entities' requests are issued from (Clifton 2012).

The taxonomy in Table 5-2 introduces the measurement attributes belong to web analytics context (Kohavi et al. 2002). The taxonomy excludes other attributes such as those discussed in business analytics (Holsapple et al. 2014), which support broader aspect than customer centric application. Business analytics provide better insights particularly from operational data stored in transactional systems to inform sales, marketing, price optimization and workforce analysis (Kohavi et al. 2002). The data are usually collected offline by the executive staff in a company (Shung and Junyu 2012) or an e-commerce platform (Kohavi et al. 2002).

This section confirms the usage of software analytics for product planning, but that it is yet to be understood how the measurements would be used for product planning evolution. These are the aims of the current study.

3. Research Design

To achieve the discussed aims, we designed an inductive study based on product managers' interpretations of analytics for product planning. We explored the following research question:

RQ: How are analytics used for planning product evolution?

To answer the research question, we conducted an interview-based survey with the purpose of identifying the relation between analytics and decisions of product planning. We performed data collection using semi-structured phone interviews. For the interviews, we initially designed the questionnaires, but we also asked the interviewees about their motivations for the provided answers. To avoid disadvantages of telephone survey related to lack of visual material and avoid complexity, the screen of the interviewer's computer that presents the questionnaire was shared with interviewees through web-based screen sharing applications.

Samples: We asked a well-established consultancy company in software product management to introduce experienced SaaS product managers in a wide variety of SaaS contexts. We selected 17 product managers from 3 micro, 4 small, 7 medium, and 3 large companies. The product managers managed 7 new respectively 10 already existing software products. All interviews were structured alike. The similarity of questions, homogeneity of interviewees and number of interviews could make the saturation of the interview results (Guest et al. 2006).

Designing the instrument: We designed a questionnaire in which the taxonomy of measurement attributes discussed in section 2 was a base for asking product managers how they would use analytics. The questionnaire was started with questions about context facets of the product, organization (company size and development team size) and people (role and experience). Questions about product planning formed the core of the interview, in two parts: "Planning Decisions" and "Analytics". In the first set of questions, the interviewees were asked to select a product that they have planned and are most satisfied with. Then questions were asked about the planning decisions that the interviewees usually take for the selected product. Later on, the interviewees were asked to rate the importance level of measurement-categories and measurement-attributes for taking the decisions and provide comments for their reasons behind the selections.

Interviews were piloted by two product managers and two students having product planning knowledge. After initial testing and several refinements, the interviews with the product managers were scheduled.

Selecting and presenting the results: We recorded the interviews by getting permissions from interviewees for the sake of future reference and transcribed for qualitative analysis of their argumentations. From the selected applications, 4 were "Business oriented", 7 were "Consumer-oriented" software and 5 were "information display and transaction entry". 41.2% of the products were new products, and 58.8% of the responses were evolutionary products. The distribution of interviews among different application magnifies the difference of product characteristics on interview results.

Analysis method: We used inductive content analysis approach (Elo and Kyngäs 2008) for analyzing and coding the argumentations of the interviewees. In the first step of the analysis, we selected a unit of arguments, tagged with the headings describing the argumentations' concepts for the role of analytics, and repeated the process for all arguments. In the next steps, we grouped the headings in two rounds to reduce the number of similar categories in each round. The categorization provided a mean of interpreting the phenomenon, increasing understandability, and facilitating decision making ability (Elo and Kyngäs 2008). At the end of the content analysis, we performed abstraction, which led to general descriptions and further discussions based on the categories. During the process, initial codes were gradually improved to form the final codes.

4. Analysis and Results

4.1. A Model for Analytics-based Product Planning

By the analysis of interviewees' argumentations, we could conclude that product managers use *analytics to interpret the product goals while the analytics are constrained by both product characteristics and product goals.* This relation has been illustrated in Figure 5-1.

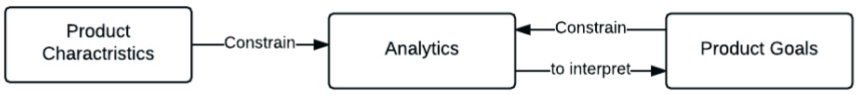


Figure 5-1. A model for anlaytics-based product planning

For building and evolution of a product, product managers define product goals aligned with the companies' business goals. The essential goal of a product is to ensure that a product is built to deliver business values to a specific set of customers and meet important business goals of companies.

The analysis of interviewees' argumentations showed that product managers did not recognize some analytics useful for specific characteristics of a software product. In another word, product characteristics limit the scope of using analytics. Table 5-3 in Appendix provides a list of product characteristics and corresponding supportive quotes about constraining analytics. As an example, the application type filters and constrains the applicable measurements:

> *"For our specific product, error and response time could be used, but others healthiness measurements did not have a role in our intranet-based product."*

Coding the argumentations clarified that analytics can be used to interpret products' goal in terms of assisting product manager to evaluate how far product goals are achieved. These products' goals might also constrain the analytics. Table 5-4 in Appendix outlines the interviewees' interpretation of analytics for product planning. The extracted codes for product goal characteristics (i.e. the left column of Table 5-4) reveal that product managers mostly addressed a dimension of product quality as a goal. "User satisfaction", "customer satisfaction", and "freedom from risk" are quality in use attributes in ISO/IEC 25010. The usability, functional suitability, maintainability, reliability, and performance efficiency codes are static and dynamic properties of software products in the quality model of ISO/IEC

25010. Such analytics support product evolution decisions from the technical perspectives.

Also, extracted code “market positioning” for product goal characteristics (i.e. the left column of Table 5-4), introduces a business goal (Clements and Bass 2010), to be interpreted by analytics. Such goals complement the technical evaluation of the product to give 360-degree view to the product manager for taking decisions (Ebert and Brinkkemper 2014).

Product managers define product goals alongside with business goals considering inputs from stakeholders. So analytics can point out to the level that a product goal has been achieved. On the other hand, the product goals can constrain analytics and specify which measurements have more or less value to achieve the desired level of the goals:

> *“For referral source measures, if I can find out in what segment the user belongs to, and then it is very important. If from the measures, I can find out from which country they use it, it is mostly less important.”*

The example indicates that extracting statistics about user’s segment from referral source attributes is valuable and can be interpreted toward a product goal, while other statistics of referral sources might not be valuable for this case. For all codes, although interviewees’ argumentations were not available to support both interpretations and constraints, the logical relations between interpreting product goals and constraining analytics can cover the argumentation shortage:

> *“Click steam is important to see the sequence of clicking to track the usage and see do the users follow the pattern in a right way or not.”*

The example illustrates that high level of click streams might interpret a good level of user satisfaction for the feature and can strengthen the quality of the feature. Logically it is evident that achieving user satisfaction wishes to have information about click streams, which strengthen the constrained relations.

4.2. Validation of the Model

The model in Figure 5-1 was validated by examples of product managers' experiences. We mapped argumentations of product managers (i.e. interviewees) for different groups of products to the model. The mapping helped us to check whether the chains of arguments can support the model. The products that interviewees selected during the interviews belonged to three product types: "Consumer-oriented software", "Business oriented" and "Information display and transaction entry". For each product type, one interview was selected to show how the shifting from constraining the analytics to interpretation of the product goal is performed. Table 5-5 in Appendix presents three examples of different products. The following example shows how argumentations of an interview (first row in Table 5-5) can support the proposed model.

Based on the characteristics of a mobile application, "referral source is not important [analytics] because users are from all over the world". "Dos and worm attacks are not important [analytics] in an iPhone application" but when the product is mature, the other "product healthiness statistics are extremely important because having errors and bad usability makes it hard [for users] to understand a feature". By collecting data about product healthiness "The errors [analytics] can be seen very quickly and repaired in each month release". So product manager will monitor analytics to find out error and take an action toward a healthy product. Having a healthy product will facilitate the customer benefit goal.

In this example "mobile application" is the product with specific characteristics, "referral source, Dos attacks and worm attacks" are analytics and "customer benefit goal" is the product goal. The relations between product characteristics, analytics, and product goals could confirm the relations defined in Figure 5-1. Similarly, the other argumentations can also confirm the defined relations in the model.

5. Discussion

In this paper, we contribute to creating a model for understanding how analytics are used for planning of a software product. The study introduces a new perspective for product planning by applying analytics. Analytics are filtered based on product characteristics and product goals. The analytics are interpreted to evaluate the level of product goals' fulfilments. The evaluation enhances a product manager's intuitions to help to find out the rationales for his decisions. Deviation from the product goal requires an action that reflects a new decision in the product plan (Kittlaus and Clough 2009).

The results have implications for research on understanding the relations between product characteristics, analytics and product goals for supporting product evolution. The results have also implications for product managers of software vendors on interpreting analytics to use data science as a basis for decision supports of product planning. In Figure 5-2, we propose a product manager to carry out a chain of activities to take planning decisions for product evolution by the supports of analytics.

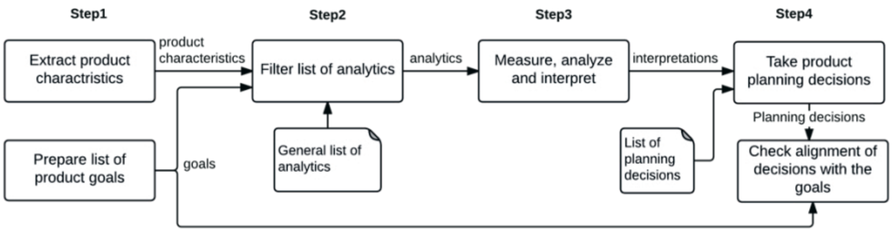

Step1
Step2
Step3
Step4
Extract product charactristics
product characteristics
Filter list of analytics
analytics
Measure, analyze and interpret
interpretations
Take product planning decisions
Planning decisions
Check alignment of decisions with the goals
Prepare list of product goals
goals
General list of analytics
List of planning decisions

Figure 5-2. Suggested activities for product managers to support planning decisions and product evolution by analytics

In step 1, the product manager prepares a list of goals corresponding to the candidate product. The study showed in a SaaS-based product, most of the product managers set quality goals with the focus on quality-in-use (ISO/IEC 25010). In this study product managers looked for acceptable perceived experience of use (efficiency), acceptable perceived results of use (effectiveness), acceptable perceived consequences of use (Freedom from risks) and the customer's satisfaction in a particular context of use (Herrera et al. 2010). Quality of services and marketing goals were also on the list of goals, with lower priority than the quality-in-use goals.

In step 2, from the general list of analytics (i.e. created using a general list of measurement attributes such as Table 5-2), the product manager excludes those with less importance based on the defined product characteristics and goals. The study showed some of the factors that constrain the analytics for product planning. Product characteristics such as product's context, features, users, platform, network type and maturity constrain the analytics for product planning. Also, product goals such as managing the quality of product, managing market positioning and organization grows can constrain the analytics. Although few goals were discussed by the interviewees, it is not a big deal to generalize to different goals such as growth and continuity of the organization, meeting financial, personal objectives, and etc. (Clements and Bass 2010).

In step 3, the included analytics are measured, analysed, and interpreted to provide required information and inform the product managers' decisions. The alignment of the decisions with the product goals is investigated in step 4. Argumentations of interviews showed, product managers usually benefit from analytics about product and feature usages, which supports goals corresponding to functional suitability and usability. Product healthiness analytics support performance efficiency, reliability and security goals. The result is in the same direction with the study that recognized feature use, product use, response time, users, error and downtime as the most preferred measurements for planning, despite planning decisions' types (Fotrousi et al. 2013). To create, remove, or enhance a feature, the data trends provide a broad view of requirements or feature desirability in the current or even future time and clarify how these changes can impact the product's goal. Comparing the corresponding measurements' impacts on the defined goals can prioritize features. This impact can support both reactive and proactive planning for an evolution of the product.

The chains of interrelated activities explained in step 3 are mapped to the measurement information model defined in ISO 15939. We propose to enhance the model by adding a box for product goals with two outgoing arrows: One to *constrain* measurement attributes and one to *support* the information needs. The enhancement would adapt the ISO 15939 to support product evolution using analytics.

The proposed model in Figure 5-1 is not specific to product planning of a traditional software development, but the model may support planning of products using modern development approaches (Olsson and Bosch 2014) such as an agile development, continuous integration, and continuous deployment. In such approaches instead of listing the product goals in the beginning (i.e. refer to step1), sub-goals of the corresponding iteration are identified instead. However, for the iterations that do not release a software product or prototype, analytics approach is not applicable. Because the prerequisite for using runtime analytics for product planning is to have a software prototype or product.

The study was limited to 17 answers of product managers experienced in SaaS-based products. However, the stratified sampling ensured the results are from the variety of product managers. Although the study focused on analytics of SaaS-based products, the model in Figure 5-1 could be generalized to the other application domains, by considering that meaningful analytics may vary in different categories of products. For example, *Throughput* measurement does only make sense in networked-based applications. Furthermore, another limitation was due to the choice of product managers for focusing on roadmapping decisions. More detailed study of portfolio management and release planning decisions may reveal other constraints on analytics in future. It is also valuable for researchers to know which measurements support each product goal and how the product manager may prioritize the measures, which we propose as future work.

6. Conclusion

Products are the artefacts to satisfy the customers' needs, and hence product managers require bringing the voice of market and customer to the product planning processes, where this happens effectively through a data-driven endeavour of sensing and understanding the requirements. Different types of analytics assist a product manager in product planning, where each might be gathered through a different channel and process. SaaS-based product delivery facilitates gathering a new range of detailed, usable and real-time product-use data. Measuring and analysing the data to support product-planning decisions are targeted by analytics.

This study introduced two taxonomies as inputs for the other parts of the study: A taxonomy of SaaS-based measurements in categories of two dimensions: “Product”, “Feature/content”, “GUI Elements” in the first dimension, and “healthiness”, “usage”, and “context” in the second dimension. The second taxonomy was related to planning decisions taken in portfolio management, roadmapping and release planning.

To present how analytics assist product managers and contribute to product planning, an interview-based survey was conducted with professionals in the product management area by focusing on roadmapping decisions since the interviewees were experienced more. Through the interview-based survey, the justifications of interviewees for assigning a value to a measurement show that both product characteristics and product goals constrain analytics, while it is interpreted to product goals. In the other word, product characteristics and product goals specify which analytics can assist product managers in achieving the product goals.

The findings helped us to propose an analytics-based model. Some parameters such as product maturity, users, network type, context, and technology change the scope of analytics usefulness for product planning. Analytics can be motivators for product managers to achieve goals for market positioning, meeting quality-in-use (i.e. customer and user satisfaction) and improving product quality (usability, functional suitability, maintainability, reliability and performance efficiency). Therefore, even limited list of analytics will be helpful to gain good support for taking planning decisions aligned with the product goals. In the case that analytics shows any deviation from the product goal, the product manager takes a constructive decision to prevent its occurrence or, at least, decrease negative effects. The analytics-based model can be used in various application domains rather than SaaS, when collecting the customized analytics for a particular domain is applicable.

Acknowledgments. Parts of this work have been done with the support of the SUPERSEDE project funded by the European Union's ICT 2014 under grant agreement no 644018.

Appendix

This section shows the detailed qualitative analysis.

Table 5-3. Constraining analytics*

Product Characteristics	Constraints
Product maturity	“When you are creating an immature product, it is hard to base your decision based on these kinds of statistics. Instead of analytics for creating decision for an immature product, we create a prototype and test the prototype. But for tuning functionality and enhancing, these statistics can have benefits.”
	“From a second release to third release, definitely analytics can be helpful. Product-use [measurement] affects their allocation of feature in third release. But not from first release to second, because first release is mainly about how to build a product.”
Product users	“Referral source attribute is not important because our users are from all over the world as they use their mobile phone.”
	“End users are within some specific organizations so statistics about referral sources are not important.”
	“Statistics about new user are not important because we are dealing with available users, not new users.”
Being Web based	“Technology and channel [measurement] is very important because the product is a web-based tool.”
	“Technology and channel data is less important. We need to support all browsers and cover related technology as it is a web-based product.”
Network type	“For our specific product, error and response time could be used, and others [other healthiness measurements] did not have a role in the intranet-based product.”
Product context	“Dos and worm attacks are not important in an iPhone application.”

Product Characteristics	Constraints
Product technology	“Technology and channel data are less important. We have to support all browsers and cover related technology as it is a web-based product.”
	“Inside our organization it is clear which OS or browsers the product has to work with, so we did not have too many challenges about it [Technology and channels measurement attributes]”
Product features	“Language attribute is not important. Our product only supports English language, and there is no different to know what languages have the users.”

*: Words are given in brackets (i.e. []) has not directly mentioned in quotes and was added to make the interviewees quotes clearer.

Table 5-4. Examples of analytics interpretation for product goals and the constraints that a product goal provides for analytics

Product goal characteristics	Interpretation	Constraint
Market positioning	“Statistics about campaign are important because they show how efficient various marketing campaigns are in bringing visitors to be customers.” “Referral source measurements can be interesting as we can learn about the structure of the market and then they can map it to the feature use, by that make it an input for prioritizing features for further development. So in combination with other studies of a market, it is important but alone and in isolated manner.” “Our goal is to increase web users, if product use is not too many then action should be taken to find the reason..”	“For referral source measures, if I can find out in what segment the user belongs to then it is very important. If from the measure I find out from which country they use it, it is mostly less important.” “Referral source is not importance since we sell product to an organization not end users. So they do not care where the customers are coming from.”
Customer Satisfaction	“Our main role is to create customer benefit to the product and give them functionality that is useful. For example by analytics, finding errors can be seen very quickly and repaired in each month release.”	“In our product, it is good to create more customer benefit which are got from an interview with customers and customer feedback from their service organizations. If we agree on prioritizing feature, the statistics are not useful for them.“

Product goal characteristics	Interpretation	Constraint
Functional Suitability	“Referral source measure attributes are important because you can help to adapt User Interfaces.” “Statistics in Technology and channels are important because we do not want to support all versions and will support technologies that are used more.” “Technology and channels statistics are very important-Depending on which mobile they have accessed from they have to provide a service according to that.”	“Technology is a tricky category, what do you mean by technology? Technology that used for development, or technology that is related to users. They are different with each other. For development part the analytics is not important, although for user side that plays important role.”
Reliability	“Product healthiness [analytics] is very important. If we cannot achieve desire reliability and performance we can go home.”	“All healthiness measures are important, especially error, people do not accept faulty product and error.”

Table 5-5. Examples of shifting from constraining analytics use to interpretation of analytics for product planning

Product characteristics	Constrain (by product characteristics)	Interpretation	Product goal	Constraint (by product goal)
Social ERP (Business oriented product)	“Exit and entrance feature is mostly good to know when you have a product like a website. For other product it might be different entrances and exits and might not so differ to each other.” “Referral source measurements are not so interesting. I think they are mostly useful for websites, like online shopping to know the source of customers. For us, the current user’s location is clear.”	“Quality adds value to the product. If not, you [i.e. your products] are definitely dead. Faulty product ends in no user satisfaction. So, It's good to know before lose all users. “ “Feature measurements Provide a good picture of interesting features”	“Planning a high-quality product is that makes users satisfied is important.”	*“Product user is very important to monitor the popularity level of product during time period.”*

Product characteristics	Constrain (by product characteristics)	Interpretation	Product goal	Constraint (by product goal)
SaaS-based Knowledge Management (Information display and transaction entry product type)	“End users are inside some specific organizations, so referral source measurements are not important for us. Also, inside each organization it is clear which OS or browser are available, so we do not have too much challenges about it.”	“Lead users have special roles in patterns related to gathering tacit knowledge in the organizations. So, it is important to understand who are the lead users to target specific users. So, user classification based on their activities on the product is useful.” “Depth of use analytic helps us to understand that users are involved with the product and do not have random visiting. “	“The product supposed to grab tacit knowledge in the organization so it was important that adequate number of users would engage in different parts of the system.”	“Feature use is very important because it shows which parts of the system the users are engaged.”

PART 4

Combining User Feedback and Monitoring Data from Software in Use

Chapter 6 : Quality Requirements Elicitation Based on Inquiry of Quality-Impact Relationships

[This chapter is based on : ***F. Fotrousi***, *S. Fricker, M. Fiedler (2014). "Quality Requirements Elicitation based on Inquiry of Quality-Impact Relationships", 22nd International Conference on Requirements Engineering (RE), Karlskrona, Sweden: IEEE, pp: 303-312. DOI: 10.1109/RE.2014.6912272.]*

Abstract

Quality requirements, an important class of non-functional requirements, are inherently difficult to elicit. Particularly challenging is the definition of good-enough quality. The problem cannot be avoided though, because hitting the right quality level is critical. Too little quality leads to churn for the software product. Excessive quality generates unnecessary cost and drains the resources of the operating platform. To address this problem, we propose to elicit the specific relationships between software quality levels and their impacts for given quality attributes and stakeholders. An understanding of each such relationship can then be used to specify the right level of quality by deciding about acceptable impacts. The Quality-Impact relationships can be used to design and dimension a software system appropriately and, in a second step, to develop service level agreements that allow re-use of the obtained knowledge of good-enough quality. This paper describes an approach to elicit such quality–impact relationships and to use them for specifying quality requirements. The approach has been applied with user representatives in requirements workshops and used for determining Quality of Service (QoS) requirements based the involved users' Quality of Experience (QoE). The paper describes the approach in detail and reports early experiences from applying the approach.

Keywords

Requirement elicitation, quality attributes, non-functional requirements, quality of experience (QoE), quality of service (QoS)

1. Introduction

Quality requirements are an important class of non-functional requirements (Glinz 2007). They concern software system attributes such as functional suitability, performance, reliability, usability, security, and portability that are important for achieving stakeholder goals (Boegh 2008). The satisfaction of these quality attributes determines whether the software system meets the goals of its stakeholders or whether the system has a negative impact for these stakeholders (Chung et al. 2000; Haigh 2010)

Meeting the right level of quality is important to balance benefits and cost (Regnell et al. 2008). The quality of a software system needs to be at least as good as to make the software useful and competitive but should not be excessive to avoid cost and unnecessary use of resources. Insufficient quality leads to disappointment and consequent churn when stakeholders decide to abandon the software solution and adopt alternatives instead (Kilkki 2008). Excessive quality may lead to an unnecessarily expensive design of the software system (Bass et al. 2012), to unnecessary consumption of resources needed for operating the system (Jung et al. 2010), and to trade-offs where other quality attributes suffer (Braz et al. 2007).

To address the problem of finding the level of good-enough quality, the relationship between software quality and the impacts of such quality for the stakeholders of the software system needs to be understood. As demonstrated for the Quality of Service (QoS) of a telecommunication network and the Quality of Experience (QoE) of the network users, a quality–impact relationship can be developed empirically by setting quality levels of a given quality attribute and measuring the reaction of the stakeholders that were exposed to these quality levels (Fiedler et al. 2010).

This paper describes how to use quality–impact analysis for eliciting requirements about good-enough quality of a software system. The proposed method guides the elicitation of the quality–impact relationships and explains how to use the gained insights to specify quality requirements. The method delivers empirical evidence for a specific software system that is more reliable than generic expert opinion. The evidence pertains to the features that were investigated and the stakeholders that were participating in the requirements inquiry, thus is adequate and relevant for decision-making about that software system's quality requirements.

The paper describes the proposed quality–impact elicitation method in depth. It gives details about the key ideas of the method and explains how to tailor the method depending on the investigated quality characteristics, the stakeholder goals impacted by these quality characteristics, and the instruments that the investigator is able to apply. The paper provides an example of how the method is applied in practice by reporting about its use in a real-world software development project.

The paper is structured as follows. Section 2 reviews existing work and motivates quality requirements elicitation based on quality–impact relationship inquiry. Section 3 describes the method in-depth. Section 4 describes how the method is applied and reports the lessons-learned from such method application. Section 5 compares the method and the obtained results with related work. Section 6 concludes.

2. Related Work

According to ISO/IEC FDIS 25010, the quality of a system is the degree to which the system satisfies the needs of its stakeholders. The determination of whether a system exhibits the desired quality characteristics is not straightforward, however. In contrast to functional requirements, many quality requirements do not have a sharp boundary between satisfaction and non-satisfaction. Instead, they are gradually satisfied (Glinz 2005), thus called soft requirements (Irvine and Levin 2000).

The softness characteristic of implies that the right level of desired requirements quality needs to be identified during requirements engineering (Regnell et al. 2008). Each such quality level has its own specific costs and benefits. High quality levels are considered more costly than low quality levels because more expensive designs or approaches to provision of the software service need to be chosen to implement the requirement. In a similar vein, increase of the quality level implies increase of the benefits generated by the requirement. A product that is considered useless because of too low quality becomes useful or even competitive with increased quality. Too much quality, however, is considered excessive thus not adding any value for stakeholders despite quality improvement. The trade-off between cost and value impacts is a basis to determine the desired quality level and specify the requirement in a quantified manner (Gilb 2005; Jacobs 1999).

Goal models have been proposed to elicit quality requirements (Antón and Potts 1998; Chung et al. 2000). Such models allow identification of needs for improving, increasing, or keeping the level of the quality characteristics of a software. To support systematic identification of goals and qualities within a given domain, ontologies have been developed and used to support requirements elicitation (Souag et al. 2012; Wang et al. 2010). The means-ends relationships that are an inherent part of a goal model make the impact of such a quality requirements explicit (Cysneiros and Sampaio do Prado Leite 2004; Herrmann and Paech 2008). The goals that are enabled by such a decision are used as a rationale that motivates the quality requirement.

Unfortunately, goal models are of limited help eliciting appropriate levels of quality. Goal models help identifying the quality characteristics that are perceived relevant by stakeholders, and the means-ends relationships connect these qualities to the impact that is desired by the stakeholders. However, they do not guide a requirement engineer in how much of a desired quality is good enough. One of the key limitations is that the goal models do not relate a given quality level to a given level of impact beyond the coarse-grained levels of a requirement being denied, weakly denied, undecided, weakly satisfied, and satisficed. In addition, the application of goal models does not deliver the information needed to quantify a quality requirement, thus make its satisfaction measurable with attributes such as scale and meter (Jacobs 1999).

Several supporting elicitation methods have been proposed for requirements elicitation (Pohl and Rupp 2011). These include the use of questionnaires, interviews, workshops, creativity methods, storyboards, use cases, role-plays, and prototyping. Review of prototypes has been particularly effective in identifying usability concerns and refining user interaction design to reach user acceptance (Rettig 1994). The construction of such prototypes allows a development team to capture assumptions about desired software characteristics and to validate these assumptions, for example by reviewing them as implementation proposals with concerned stakeholders (Fricker and Glinz 2010; Fricker et al. 2010).

The supporting elicitation methods provide limited support for the determination of good-enough quality levels because of their generality. Any question can be asked in a questionnaire or interview, any topic explored in a workshop, and a multitude of design decisions be captured with storyboards, use cases, role-play, and prototypes. Guidelines that have been proposed to identify quality requirements (Hassenzahl et al. 2001; Kusters et al. 1999) target the discovery of quality, but do not help in determining measurable levels of quality. The requirements engineer is thus left with his intuition or experience for asking the right questions (Doerr et al. 2005). The use of experience, however, is risky as the levels for good-enough quality may change between different software products and product-usage contexts.

To enable requirements engineers to determine appropriate levels of good-enough system quality, we were studying the field of telecommunication. In particular we were looking for approaches that allow the requirements engineer and the system stakeholders understand the meaning of a given level of quality, for example in terms of how the quality level affects the degree of stakeholder satisfaction. In the field of telecommunication, substantial work has been performed for understanding how to measure degrees system quality and how a given degree of system quality affects user attitude (Fiedler et al. 2010).

For a telecommunication system, Quality of Service (QoS) requirements are stated that concern system performance, availability, and capacity (Wang et al. 2010). Often these requirements are agreed in a service level agreements (SLA) between the system customers and the supplier(Kittlaus and Clough 2009). User satisfaction is expressed as Quality of Experience (QoE) and refers to the "degree of delight or annoyance of the user of an application or service" (Le Callet et al. 2012). It has been shown that a system's Quality of Service affects the user's Quality of experience (Fiedler et al. 2010). Too little user delight and too much user annoyance leads to churn, thus users that try to look for alternatives and try to avoid using the system under consideration.

The knowledge of how QoS is related to QoE has not been translated into requirements engineering methodology yet. In particular, it is unclear how to exploit the relationship between QoS levels with QoE levels in the inquiry of software systems requirements. Also needed is an explanation of how to apply the specifics of the QoS-QoE relationship on the determination of good-enough quality for any system quality attribute and for any important stakeholder need that is impacted by the possible quality levels.

3. Quality-Impact Inquiry

This paper proposes a method that we call Quality-Impact Inquiry to address the so far unsatisfactorily solved problem of determining adequate levels of quality. As required from a solution proposal, we have explained why a novel method was needed, specify the principles and steps of the method, and describe how to apply it (Wieringa et al. 2006). To demonstrate that the method is sound, we go a step further than required from a solution paper and report about a preliminary validation that we performed with a real-world software development project. The paper describes the method in sufficient depth to enable replication in practice and further validation research.

The Quality-Impact Inquiry method is based on the principles outlined in our earlier work about the generic relationships between Quality of Service and Quality of Experience (Fiedler et al. 2010). These principles have been translated into a software requirements engineering context by integrating it into an inquiry-based requirements analysis process (Potts et al. 1994) and combined with prototyping, questionnaires, and workshops as supporting methods for collection of quality measurements and stakeholder opinions. During the workshop, stakeholders are exposed to requirements engineer-defined quality that has been implemented in the prototype and questioned about their perceived quality impact. The correlation between quality measurement and stakeholder opinion is analysed and used as decision-support to determine and then specify good-enough requirements quality.

The Quality-Impact Inquiry method adapts the inquiry cycle of requirement analysis (Potts et al. 1994) as follows: the documentation phase is adapted to implement a prototype using a set of accepted requirements described the desired system and collects quality attributes during stakeholder actions. The three elements of requirement discussion phase including questions, answers and reasons are supported by the questionnaire elicitation. Finally, the results from the former phases contribute to either freeze or change requirements in the evolution phase.

Figure 6-1 gives an overview of the Quality-Impact Inquiry process. The remainder of this section describes the generic Quality-Impact Inquiry method and how the method may be tailored. The ensuing section describes how the method has been applied in real-world projects and reports about early lessons learned.

3.1. Inquiry Process

Figure 6-1 gives an overview over the process that characterizes the Quality-Impact Inquiry method. The process contains four steps: preparation, measurement, analysis and decision-making. It is applied iteratively until enough evidence has been collected to decide about what good-enough quality should be for a quality attribute under investigation.

1) Preparation: During the first step, Preparation, the materials needed for allowing stakeholders to experience the quality characteristics under investigation are prepared. The work includes the preparation and documentation of a prototype, the formulation of a questionnaire, the recruitment of stakeholders for participation in a workshop, and the scheduling of the workshop.

In the proposed method, quality impact is measured subjectively through a questionnaire. The quality impact is also affected by a real value of quality that is measured objectively (Brooks and Hestnes 2010) and automatically using a prototype. Therefore, a list of valid quality requirements is identified from SRS document that is relevant to one feature or a group of features (f) and presented as pairs of quality attribute and value:

$$Q = \{ (q_{att} , q_{val}) \mid f \} \quad (1)$$

As an example, in SRS, a non-functional requirement can be stated as "response time should be less that 2 s". "Response time" is the attribute and 2 s is the value.

The software might be in a preliminary release (i.e. pre-alpha, alpha and beta testing), a candidate release close to a final product/service, or even a released product ready for an evolution. Preparation of artefacts including a prototype from a software feature(s) and a questionnaire about their quality is the pre-requisite to run the method. The stakeholders experience the software and then answer the questionnaire. Data that are collected from the software use and the questionnaire are analysed to evolve quality requirements in the software specification document (SRS) if needed.

Based on the quality attributes, the prototype is tailored for the feature(s) f to support measurement of Q. The questionnaire will be tailored using Q to collect quality impacts of feature(s) f relevant to user list U:

$$U = \{u\} \quad (2)$$

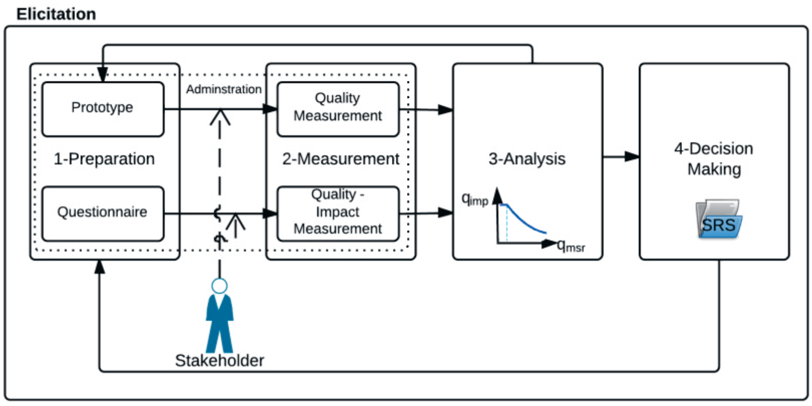


Figure 6-1: Quality-Impact inquiry method

Then, scenarios for data collection, and software guidelines to be followed by users are prepared in this step. Translating the questionnaire to the user's mother tongue is another action that might be required.

2) Measurement: During the second step, Measurement, a workshop is performed with the aim of collecting quality measurements and user feedback. During the workshop the stakeholders experience predetermined qualities by utilizing the prototype according to a pre-defined script. During the use of the prototype measurements are taken about the quality that the stakeholders experienced. After the use of the prototype, the prepared questionnaire is administered to collect stakeholder opinions about the impacts of the perceived quality.

While the users are using the application through clients such as a smartphone or a PC, quality values q_{msr} (i.e. q_{msr} is a q_{val} relevant to q_{att} for feature(s) f) are quantified by function m, automatically using analytical tools, server log generators or piece of codes embedded in the software.

$$q_{msr} = OP(\ m(\ q_{att} \mid u, f\ )\ ) \mid OP \in \{\ MIN \text{ or } MAX\ \} \quad (3)$$

The function captures the worst value of measured quality attributes in different actions of a user for the given feature(s) f, depending on whether the quality has a success or failure measure characteristic

(Fiedler and Hoßfeld 2010). For a success measure such as availability, the higher value of the quality attribute shows better quality but for a failure measure such as response time, a higher value of the quality attribute shows worst quality. Therefore, minimum or maximum value of each case would be the candidate value for measured quality.

Another source of measurement is the questionnaire designed to translate the quality impacts q_{imp} (i.e. q_{imp} is a q_{val} relevant to q_{att} for feature *f*) into scored values provided by users. In the questionnaire, users are typically asked to provide ratings,

$$q_{imp} = s(q_{att} \mid u , f) \quad (4)$$

and rationales in forms of comments that explain their ratings:

$$comm = c(q_{att} \mid u , f \quad (5)$$

Furthermore, the questionnaire asks users to rate "quality in use" attributes such as satisfaction as a sub list of quality attributes:

$$Q_{inUse} \subset Q \quad (6)$$

The quality impact is translated into a discrete value that is scaled using scores such as Mean Opinion Score (MOS) (ITU-T 2003).

3) Analysis: During the third step, Analysis, the quality measurements are correlated with the stakeholder opinions about quality impact. This step involves application of statistical analyses based on data that has been collected during the measurement step in the ongoing and previous Quality-Impact Inquiry iterations. The analysis can also be enhanced through a-priori knowledge of the generic nature of the studied Quality-Impact relationships.

The relation between the measured quality (q_{msr}) and quality impact (q_{imp}) will be identified through a regression analysis, similar to correlation analysis between QoE and QoS (Kim et al. 2008a; Minhas and Fiedler 2013). The regression function is calculated for a feature f and quality attribute qatt:

$$\hat{q}_{imp}(q_{msr}) = r(q_{msr} \mid f , q_{att}) \quad (7)$$

Different regression functions for the relationship including linear, logarithmic, exponential and power have potential to be candidate, however the analysis compares the regression function and matches the best one.

Then, an estimation of quality value for a given quality impact is calculated by the inverse function of the regression model:

$$\hat{q}_{msr}(q_{imp}) = r^{-1}(q_{imp} \mid f , q_{att}) \quad (8)$$

The output of the analysis proposes a list of quality values for different quality impacts including maximum quality impact.

If the Quality-Impact analysis does not provide enough data for a mature analysis, some changes on the prototype are applied to change the quality values artificially. The looped arrow from analysis box to prototyping box in Figure 6-1 provides possibilities to achieve enough data for investigating impact changes and perform more reliable analysis than the analysis of less data points.

4) Decision-Making: During the fourth step, Decision-Making, the analysis results are used to decide about acceptable and desired levels of quality of the investigated quality attributes. The decisions are recorded in the software requirements specification. The step concludes with decision-making about whether to add inquiry iterations and how the parameters of these ensuing inquiries should be adapted for best improving the knowledge about good-enough quality.

The decision-making process selects suitable quality value from the evidences and decides whether to evolve the value for the relevant quality requirement in the SRS document.

This process identifies maximum applicable quality impact considering technical feasibility, product strategies, and limitation of resources to achieve the relevant quality value, and then applies the decision-making function.

Decision-making is a function of parameters including estimated quality value for maximum impact ($\hat{q}_{msr}$) of a quality attribute, the value of relevant non-functional requirement (q_{SRS}), the list of rationale for the quality attribute rating (*comm*) beside all quality-in-use ratings (Q_{inUse}), to interpret whether the current quality fulfills the users acceptance.

$$q_{new} = \{ g(\hat{q}_{msr} , q_{SRS} , comm , Q_{inUse} \mid f , q_{att}) \} \quad (9)$$

This function defines a new value for the quality attribute. The decision-making will be performed for all quality attributes in Q.

3.2. Method Tailoring

There are a wide variety of variation points to adapt the generic Quality-Impact Inquiry process. The variations are needed to be flexible enough to adapt the process to specific requirements engineering constellations. Table 6-1 gives an overview.

4. Real-World Example of Method Application

4.1. Example Application

To demonstrate how to implement the method in practical situations, we present here the results and lessons-learned of an early validation that we have done in a real-world project. We applied the method for a Diabetes Smartphone Application that will be used by diabetes patients to take blood glucose measurements, to plan insulin injection, and to send the collected observation history to a diabetes specialist for consultation. We evaluated the Quality-Impact relationships for the features user authentication and observation sharing of diabetes information.

As an input to the Quality-Impact inquiry we had used a prototype that was instrumented with software for monitoring the timing of user interactions. The inquiry was performed in a laboratory and with a smart phone from the application developers with pre-loaded data. The requirements engineer, the product manager, and selected end-users participated in the inquiry workshop. The inquiry was performed with one end-user at the time.

During the inquiry, the end-user was introduced to the tasks he to be performed with the application, was given a short, tailored user manual, and then used the selected features first according to instructions and then without help. He opened the application, selected the data he wanted to share with his clinician, authenticated himself, and submitted the data. Then the authentication service requested username and password. When authenticated, the data was sent to the application server in the hospital. After the guided and unguided experiences were concluded, the end-user filled out the quality of experience questionnaire. Figure 6-2 gives an impression of the setup.

Table 6-1. An overview of variations

Variation Point	**Variants**
Software Features	Stakeholders may be exposed to different features. Quality requirements may be specific to features or the impact of quality levels be perceived differently depending on the feature.
Quality Attributes	Stakeholders may be exposed to different quality attributes. Each feature or application may have its own set of prioritized quality attributes.
Quality Levels	For the selected quality attributes, different quality levels may be investigated. The selection of the quality level should be based on information need and be guided by statistical analysis methodology.
Stakeholder Sampling	Different individuals may be invited for participation in the inquiry workshops. The selected stakeholders should be as representative as possible.
Impact Attributes	Stakeholders may be questioned about different quality impacts. Each application or feature may aim at achieving its own specific impacts.
Measurements	Different measurements may be selected to record quality levels and stakeholder impacts.
Prototyping Approaches	The simulation of different quality characteristics may require different approaches of building the quality-simulating prototype.
Impact Function	Different impact functions may be chosen the represent the relationship between a given quality attribute and its impact. We were using linear and exponential functions so far.

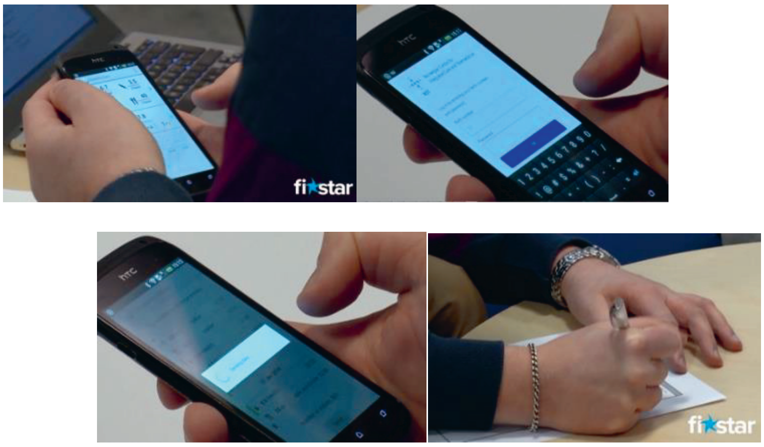


Figure 6-2. User interaction scenario with instrumented application and subsequent answering of the quality of experience questionnaire

The Quality-Impact inquiry processes was implemented for the Diabetes Smartphone Application as follows:

1) Preparation: The requirements engineer extracted relevant quality requirements from the software requirement specification document. Based on these extracts he instrumented the software with a time-stamp logger.

The requirements engineer created a short guideline to assist the end-user in using the application. It described the features to be evaluated and how the features should be used.

Based on the extracted quality requirements, the requirements engineer created a quality of experience questionnaire with generic questions about the experience, about the features and product, and about the perceived quality. For the Diabetes Smartphone Application, the quality questions were about performance, reliability, and availability. Figure 6-3 shows the questionnaire.

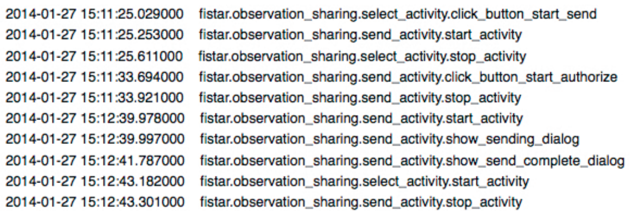
2) Measurement: The following steps describe the inquiry workshop that was performed once for each user separately.

In the beginning of the inquiry the requirements engineer welcomed the participants, defined the goals of the inquiry, and shared the agenda of the meeting.

The product manager explained the feature to be used and gave prepared guideline to the end-user.

The Experience
1. Please tell us the name you would give to the feature:
The Features and Product
2 Overall, how satisfied are you with the features you just have experienced? □ Excellent (5) □ Good (4) □ Fair (3) □ Poor (2) □ Bad (1) Please tell us why you feel that way:
3. Overall, how good is the feature according to your opinion? □ Exceptional □ Better than comparable products and features □ Good-enough □ Insufficient Please tell us why you feel that way:
4. Will you return to use the product again? □ Yes □ No Please tell us why you feel that way:
The Quality
5. The next question is about response time. With response time we mean the time when you press a button until the software does what it is supposed to do. How do you rate the response time of the feature? □ Excellent (5) □ Good (4) □ Fair (3) □ Poor (2) □ Bad (1) Please tell us why you feel that way:

Figure 6-3. Questionnaire. The last question can be replicated and adapted to any feature the requirement engineer is interested of.



```
2014-01-27 15:11:25.029000    fistar.observation_sharing.select_activity.click_button_start_send
2014-01-27 15:11:25.253000    fistar.observation_sharing.send_activity.start_activity
2014-01-27 15:11:25.611000    fistar.observation_sharing.select_activity.stop_activity
2014-01-27 15:11:33.694000    fistar.observation_sharing.send_activity.click_button_start_authorize
2014-01-27 15:11:33.921000    fistar.observation_sharing.send_activity.stop_activity
2014-01-27 15:12:39.978000    fistar.observation_sharing.send_activity.start_activity
2014-01-27 15:12:39.997000    fistar.observation_sharing.send_activity.show_sending_dialog
2014-01-27 15:12:41.787000    fistar.observation_sharing.send_activity.show_send_complete_dialog
2014-01-27 15:12:43.182000    fistar.observation_sharing.select_activity.start_activity
2014-01-27 15:12:43.301000    fistar.observation_sharing.send_activity.stop_activity
```

Figure 6-4. Extract from the log file with timestamps and activities

The end-user used the application according to the instructions. He did so twice to allow us collecting data about the learning and knowledgeable use of the feature. The application generated logs automatically and captured information from the user interaction (see Figure 6-4 for an example). In all timestamp, the time from the internal clock on smartphone was used. Log entries were created when end user requests are received and when application screen/data have been displayed. The response time extracted from the example is the duration between two-time stamps taken from the starting to the ending of an activity.

After application usage, the requirements engineer provided instructions for answering the quality of experience questionnaire. The user answered the questionnaire accordingly. The answers that were collected with quantitative scales provided data for calculating the Quality-Impact relationship. The qualitative rationale that the users gave for these values assisted us in interpreting the quantitative values.

At the end of the session, the requirements engineer debriefed the participants and thanked them for the participation.

3) Analysis: The filled-in questionnaires and time-stamp logs from all end users' interactions were the inputs for the analysis process. The end-users were satisfied with the quality as they reflected in the questionnaire Therefore the analysis of this example did not identify any deviation to update quality attributes. However the similar study was conducted in our lab where users perception of response time in downloading a webpage containing an image were collected (Shaikh et al. 2010). The analysis of the data distributions concluded a close match for a regression formula on relations between MOS and response time excluding null opinion scores:

$$\hat{q}_{imp}(q_{msr}) = 4.836 \exp(-0.15\, q_{msr}) \quad (10)$$

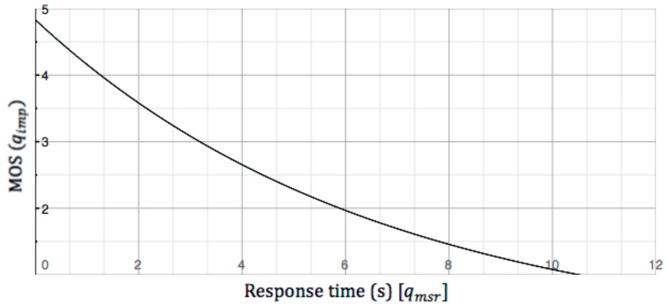


Figure 6-5. Quality impact (MOS) as a function of quality value (response time(s))

Figure 6-5 plots this regression function that shows quality (q_{imp}) as a function of quality value (q_{msr}) (Shaikh et al. 2010). The response time collected from different experiments as well as collected relevant quality impacts will plot the Figure 6-6. Taking the reverse of this function estimates quality value ($\hat{q}_{msr}$) as a function of quality impact (q_{imp}):

$$\hat{q}_{msr}(q_{imp}) = -6.67 \ln(q_{imp} / 4.836)\text{s} \quad (11)$$

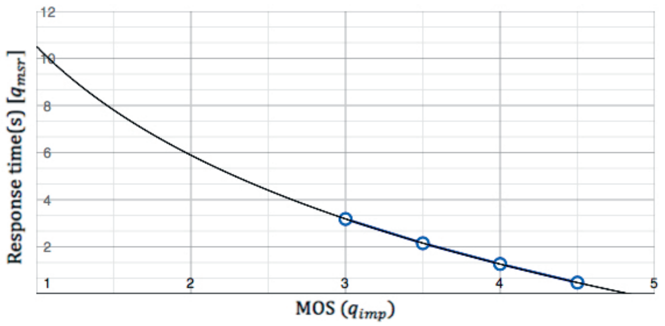


Figure 6-6. Quality value (Response time (s)) as a function of quality impact (MOS)

As Figure 6-6 plots the inverse regression function, shorter response identifies the better perception of quality and user's score. Table 6-2 estimates quality values for qimp in the range of between 3 and 4.5. This value identifies the best threshold value for a quality attribute such as response time that is sufficient for the user expectations.

Table 6-2. Estimated quality values for given quality impacts

Quality impact (q_{imp}) MOS	**Estimated quality value ($\hat{q}_{msr}$) for Response time**
4.5	0.48 s
4	1.27 s
3.5	2.15 s
3	3.18 s

4) Decision making: Decision making process involves choosing a threshold value for a quality attribute based on inputs from analysis including an estimated quality value for response time, user experiences and rationales, the list of quality-in-use as well as the value of response time defined in the SRS document.

Selecting the good-enough quality level requires trade-offs between the reaching enough user acceptance level instead of maximum level in return for gaining technical feasibility by limited resources such as cost, time and effort. Identifying maximum applicable user perception (quality impact) in each analysis is the result of such trade-offs. If quality impact 4 is recognized enough, then the estimated quality value of 1.27s will be involved in decision making process to update SRS with a good-enough quality value. Typically, the critical value for quality impact is assumed to be 3. In telecommunication area, accepted quality impact in video streaming is considered as 3.5, although the quality impact of 4 is a good choice (Khan et al. 2010).

5. Lesson learned

As shown in the example, the inquiry workshop allowed us to collect the data necessary for analyzing the Quality-Impact relationship for response time and quality of experience. The workshop lasted about 10 minutes per user. Data aggregation and analysis was concluded within a few hours. Thus the method was relatively efficient. Scalability can be achieved by working with multiple users in parallel, for example as part of a training workshop.

From the users that participated in the inquiry workshops we received positive feedback about the experience and about most of the questions we asked. However, one of the users was puzzled about perceived reliability and availability. He stated that he expected the application to work and to be available in the laboratory situation he was invited to. This shows that usage context affects the relevance of quality attributes. Some quality attributes are relevant in some contexts only. We plan to account for this feedback by extending the Quality-Impact inquiry to prolonged pilot uses of the application in the real-world contexts of the users.

On little usages of the software product could not give the full impression to users. An issue relevant to a quality attribute such as availability might not be risen in a short period of use, this is what reflected by the stakeholder in the example stated in Section 4. To reach more accurate data, a prolonged usage should be planned.

Not only quality attributes are identified in the proposed Quality-Impact inquiry method, there might be some proposals for updating functional requirements extractable from the users' comments given in the questionnaire. As an example, if the end-user could not find how to submit the blood glucose data, this could be reflected in the users' perception rating as well as provided rationale.

Training before and during the workshop provides knowledge and skills to mitigate the threats of biasing the user perception that occurred due to misuse of the feature. Distractions during the workshop should be removed to boost concentration of users in expressing their real unbiased perception.

6. Discussion

The Quality-Impact Inquiry method is a generic approach to collecting data about quality levels and how these quality levels impact stakeholder satisfaction. It builds on our earlier work that shows that a relationship between quality levels and quality impact can be established. The Quality-Impact Inquiry method extends such earlier work by describing a 4-step process that allows the requirements engineer to inquire how different levels of quality impact the satisfaction of stakeholder needs. The 4-step process is independent of the specific type of quality and independent of the specific kind of stakeholder need. Instead the method can be tailored to any pair of quality and impact measurement that are of interest for the system under consideration. A condition for such tailoring is that a relationship between quality level measurements and impact measurements can be established.

The identified level of quality impact transforms the knowledge into a judgment of good-enough quality. Good enough quality can be decided considering cost and benefit views while exposing barriers and breakpoints (Regnell et al. 2008). Product strategy decisions, competitors and learning processes are other factors that assist requirement engineer to adjust the level of quality.

The Quality-Impact Inquiry method complements existing quality requirements elicitation methods. Pairs of system quality and impact variables that should be investigated as part of requirements inquiry can be identified with goal-based inquiry methods (Chung et al. 2000; Potts et al. 1994). Means-ends relationships of prioritized soft goals that relate to system qualities, respectively to stakeholder needs, are candidates for inquiry of the corresponding Quality-Impact relationships. These candidates are used as an input to the tailoring of the Quality-Impact Inquiry method.

The Quality-Impact Inquiry method utilizes supporting elicitation methods (Pohl and Rupp 2011), in particular the use of questionnaires, prototypes, and workshops. The method combines these supporting methods into a structured process for creating and analyzing evidence for decision-making about good-enough quality. Recommendations about good practice, e.g. of how to perform an effective workshop (Gottesdiener 2002), should be followed as long as they do not interfere with the objective of the inquiry of Quality-Impact relationships that are under investigation. Side results from applying the method, e.g. the discovery of new needs or stakeholders during a workshop, should be embraced and handed over as an input to the main stream of requirements engineering work that is performed in the development project.

In a larger scale validation of the proposed method in a real-world situation various stakeholders and experienced requirements engineers are involved. To achieve trustworthy results, a specific probability is identified for considering a confident interval in which the value of quality impact lies within a specific range. Smaller numbers of stakeholders that involve in the experiment method generate wider confidence intervals since there is an inverse square root relationship between the confidence interval and the sample size. It means that to cut error margin in half, number of involved stakeholders is assumed to be four times more.

For practitioners, the Quality-Impact Inquiry method represents an extension of the requirements engineering toolset and is used for addressing the challenging problem of determining good-enough product quality. Once the relevant Quality-Impact relationships have been established, they can be reused while evolving and maintaining the application and for specifying the quality levels of comparable applications, for example in a software product line.

Quality-Impact Inquiry is not a method that is easy to apply and should thus be used by requirement engineers that are experienced in experimentation with end-users. In many practical situations, this is unproblematic. It is common to use experienced requirements engineers for critical tasks such as the development of service level agreements of software-based services (Marilly et al. 2002).

The Quality-Impact Inquiry method complements competitive analysis of product quality (Regnell et al. 2008). It allows a definition of thresholds for useful quality and excessive quality based on evidence gathered by analyzing the perception of stakeholders. In the example of QoS and QoE, the requirements engineer determines the service quality threshold by translating quality of experience judgments with the experimentally determined Quality-Impact relationship. In the real-world example described in this paper, the former was quantified with software reaction time and the latter expressed with the Mean Opinion Score. The questionnaire in Figure 6-3 shows that the relationship can also be calculated for other impacts. For example, question 3 was used to collected data about the strategic positioning of the feature according to the Quper model (Regnell et al. 2008). Question 4 allowed collecting data about the risk of churn. Any prior knowledge about the nature of the relationship, e.g. as expressed by the exponential function in (Fiedler et al. 2010), reduces the need for measurements, thus reduces the effort of Quality-Impact inquiry.

For research, an understanding of the generic relationships between levels of more types of software quality and impact is urgently needed. These generic relationships reduce the need for experimentation during real-world requirements elicitation by pointing to the functions that should be used during Quality-Impact inquiry. The characterization of the generic relationship between QoS and QoE as an exponential function (Fiedler et al. 2010) is an example of the research that is needed. Security and usability are examples of quality attributes that should be prioritized by research. The research may include investigation of what appropriate measurement scales are, e.g. of security or usability, and how a generic Quality-Impact relationship may be expressed and investigated based on scales other than the ratio scale that we used in Figure 6-5 and Figure 6-5. Also open is the development of an understanding of how the interaction of multiple quality variables, e.g. security and usability (Braz et al. 2007), can be expressed with Quality-Impact relationships, thus made amenable to requirement elicitation with the Quality-Impact Inquiry method we have presented.

The study of Quality-Impact relationships would also allow building empirical evidence for checking deeply held beliefs in the requirements engineering field. One such belief is expressed with the KANO model (Sauerwein et al. 1996). That model states that the impact of quality on stakeholder satisfaction is expressed through exponential or linear functions that describe attractive requirements, which cause delight when implemented, one-dimensional requirements, which are easily articulated, or must-be requirements, which are not obvious, but considered self-evident by stakeholders. The presented Quality-Impact Inquiry method enables practitioners to determine the exact relationships for the software products and features they are specifying. For researchers, it can be used to inform the design of empirical research studies that aim at investigating generic Quality-Impact relationships.

7. Conclusion

The paper has described an approach to quality requirements elicitation based on inquiry of Quality-Impact relationships. The method, called Quality-Impact Inquiry, guides a requirement engineer in the inquiry of good-enough software quality from the viewpoint of the appropriate stakeholders of the software system. When applying the method, stakeholders experience a prototype of a software system. The requirements engineer collects the real values of chosen quality attributes and subjective feedback from the stakeholders about perceived quality impacts. The analysis of Quality-Impact uses a regression function. The method can be tailored to pairs of qualities and impacts that are of interest for the specific software system. Systematic use of the method gives support for deciding about appropriate the quality levels. These can then be specified in a quantified manner for example by stating minimal, maximal, and expected quality in a software requirements specification (SRS) or service level agreement (SLA).

The Quality-Impact Inquiry method was applied for requirements engineering in real-world development projects. One example was shown to describe how to apply the method in practice and to report on lessons learned. We reported how we have applied the method for these requirements engineering endeavors, shared early experiences from applying the method, and have given recommendations for practical use of the method.

Future research should aim at validating and evaluating the method in further, large-scale requirement engineering situations. Moreover, future research should aim at expanding the understanding of the generic relationships between given combinations of software quality attributes and their impacts as well as how quality attributes interact with each other. The resulting knowledge will translate into a SLA and help to allow and to reuse the knowledge of appropriate quality levels. It will also help accelerating and simplifying quality requirements inquiry in real-world projects and enable research to check deeply held beliefs about how quality and impacts are interrelated.

Acknowledgments

This work has been co-sponsored by the European Commission through the FI-PPP integrated project FI-STAR under grant agreement number 318389.

Chapter 7 : FAME: Supporting Continuous Requirements Elicitation by Combining User Feedback and Monitoring

[The chapter is based on: *M. Oriol, M. Stade,* ***F. Fotrousi****, S. Nadal, J. Varga, N. Seyff, A. Abello, X. Franch, J. Marco, O. Schmidt (2018). "FAME: Supporting Continuous Requirements Elicitation by Combining User Feedback and Monitoring", 26th International Conference on Requirements Engineering (RE), Banff, Canada: IEEE. pp: 217-227. DOI: 10.1109/RE.2018.00030.*]

Abstract

Context: Software evolution ensures that software systems in use stay up to date and provide value for end-users. However, it is challenging for requirements engineers to continuously elicit needs for systems used by heterogeneous end-users who are out of organisational reach. Objective: We aim at supporting continuous requirements elicitation by combining user feedback and usage monitoring. Online feedback mechanisms enable end-users to remotely communicate problems, experiences, and opinions, while monitoring provides valuable information about runtime events. It is argued that bringing both information sources together can help requirements engineers to understand end-user needs better. Method/Tool: We present FAME, a framework for the combined and simultaneous collection of feedback and monitoring data in web and mobile contexts to support continuous requirements elicitation. In addition to a detailed discussion of our technical solution, we present the first evidence that FAME can be successfully introduced in real-world contexts. Therefore, we deployed FAME in a web application of a German small and medium-sized enterprise (SME) to collect user feedback and usage data. Results/Conclusion: Our results suggest that FAME not only can be successfully used in industrial environments but that bringing feedback and monitoring data together helps the SME to improve their understanding of end-user needs, ultimately supporting continuous requirements elicitation.

Keywords

Feedback gathering, usage monitoring, requirements, user involvement, feedback acquisition, data collection, requirements elicitation, software evolution, user feedback.

1. Introduction

Software systems face the need to continuously evolve and adapt to meet changing stakeholder needs. This requires continuous elicitation of stakeholder needs regarding the software in use (Forbrig 2017), a rapid understanding of requirements violation (Vierhauser et al. 2016), and an immediate reaction to evolve requirements (Carreño and Winbladh 2013).

Traditional requirements elicitation methods such as interviews, workshops, or prototyping can be successfully applied to software systems where the stakeholders, including end-users, are within organisational reach but are more challenging in the context of software systems used by a heterogeneous crowd of end-users in different locations. The main challenges that requirements engineers face in this context is how to continuously elicit end-user needs for the software in use, and validate if the implemented requirements are aligned with their needs (Snijders et al. 2014). In this regard, continuous requirements elicitation demands an efficient approach to elicit end-user needs remotely and in a scalable manner.

Academic literature suggests, amongst others, two suitable approaches to support continuous requirements elicitation. One approach, user feedback gathering, allows a vast number of end-users to communicate their needs for a deployed software anytime and anywhere. This approach also involves end-users who are out the organisational reach (Seyff et al. 2015). The other approach is system monitoring. For example, monitoring the user behaviour, combined with data mining techniques, have been proposed to support requirements elicitation and decision making (Guzmán et al. 2017).

Both approaches are the basis for the work done in our research. We have already evidence that they are useful to elicit new requirements (Brill and Knauss 2011b; Wellsandt et al. 2014). However, novel work argues that both approaches need to be combined to improve the requirements elicitation process(Milne and Maiden 2012). Some of this work has also included conceptual solutions and early architectural prototypes. However, to the best of our knowledge, none of the existing solutions is mature enough to be used by companies to support their continuous requirements elicitation process. Furthermore, little is known about the actual benefits of combing user feedback and monitoring in real-world scenarios.

In this paper, we present a framework that provides advanced features for the combined collection and analysis of feedback and monitoring data. To show the validity of our technical solution, we applied FAME in a German software company, which successfully integrated the framework into its requirements engineering process and are using it to support continuous requirements elicitation for one of their software products. Early evaluation results suggest that the data gathered with the help of our framework supports the software company in better understanding the needs of their end-users.

1.1. Motivating Example

In this section, we present the main challenges that a German SME (namely SEnerCon; https://www.senercon.de/) faced in the requirements elicitation process for one of their software products. To understand SEnerCon's current situation, we conducted a case study (for a detailed method description see (Stade et al. 2017)). The company develops and maintains web applications for end-users in the domain of energy efficiency management. This iESA web application enables end-users to visualise their energy consumption and guides them in saving energy (www.energiesparkonto.de).

To elicit requirements from their end-users, SEnerCon used to collect feedback from traditional feedback communication channels including email, contact forms, support hotline, and an online forum and did not use a dedicated software tool for feedback gathering. SEnerCon also had some monitoring software components installed, which were mainly used to assess the Quality of Service (QoS) and reliability of the web application, rather than helping in the requirements elicitation process. In particular, SEnerCon already obtained the QoS of its services through system logs and hardware statistics, but user feedback was the only source for SEnerCon to elicit requirements. In this situation, SEnerCon faced several challenges in their requirements elicitation process including:

Understanding the Feedback. Before SEnerCon extracted information suitable for refinement or elicitation of a requirement, the company had first to understand the feedback message. In particular, when the user feedback was incomplete, inaccurate, or unstructured, this was a cumbersome step. Sometimes, SEnerCon needed to start further investigations. This included asking the end-user questions for clarification or testing various potential scenarios where the end-user had struggled. Such additional investigations were time-consuming and, in some cases, had even annoyed the end-user when SEnerCon contacted them.

Consolidating and Prioritizing Requirements. Another challenge for SEnerCon was to estimate how many end-users might be affected by a feature request or problem, in particular, when only one end-user communicated this problem. The number of affected end-users could be a good indicator for SEnerCon to decide on the consolidation and prioritisation of a new or revised requirement.

Identifying a Proper Satisfaction Criteria. The final challenge SEnerCon were faced with when eliciting a new requirement, was to identify a satisfaction criteria correctly. Usually, the satisfaction criteria were not explicitly stated by the end-user, or it was not clear-cut in the feedback. This caused the SEnerCon team to define the satisfaction criteria based on their personal opinions.

We expect that the combination of feedback and monitoring data could help SEnerCon to solve or mitigate some of these (and further) challenges. For instance, usage data about what the end-users did before they provided feedback could be useful to clarify the end-users' intention to provide feedback or the problems they faced.

Monitoring how many end-users used a specific functionality would provide an indicator of the importance of a requirement related to that functionality. Finally, monitoring data could also provide measurements that could help to define satisfaction criteria that were not explicitly stated in feedback (e.g., the time end-users spent waiting for a task to complete before they leave the web application or provide negative feedback).

1.2. Research Objective

In this paper, we address the following research objective: To provide a unified framework capable of gathering and storing both feedback and monitoring data, as well as combining them using an ontology, to support the continuous requirements elicitation process. To achieve such a research objective we have developed a framework, named FAME, and have conducted the first evaluation together with SEnerCon, demonstrating the successful combination of feedback and monitoring data, and how such combined data can provide valuable information for the elicitation of new requirements. The remainder of the paper is structured as follows. Section 2 presents the related work. Section 3 describes our proposed technical solution. Section 4 presents our validation in a real-world context, including a description of the execution of the validation, the results, and its discussion. Finally, Section 5 presents the conclusion and future work.

2. Related Work

2.1. Feedback Gathering for Requirements Elicitation

Feedback acquisition approaches allow end-users to communicate problems, needs, and opinions while using a software product (Wellsandt et al. 2014). From such feedback data, requirements engineers can extract requirements (Seyff et al. 2015).

Several dedicated feedback tools have already been developed, which can be designed as standalone (e.g., (Seyff et al. 2014)), embedded (e.g., (Rashid et al. 2009)), or cross-platform (e.g., (Hess et al. 2012)) solutions. These and further tools aim to support and motivate end-users in communicating feedback in linguistic (e.g., free text input, category selection) or non-linguistic (e.g., rating, annotated screenshot) formats (Morales-Ramirez et al. 2015). The communication of feedback can be initiated (pushed) by the end-users (e.g., by pressing a feedback button) or by the requirement engineer who asks (pulls) for feedback (e.g., by triggering a feedback dialogue that appears to end-users in a pop-up window) (Maalej et al. 2009) but most of the research tools support only the first scenario.

Although user feedback can be a valuable data source for the requirements elicitation process, several approaches have emphasised that feedback can be hard to understand and interpret when context information related to particular feedback is missing, or when the description is unstructured (Pagano and Brügge 2013; Stade et al. 2017; Zimmermann et al. 2010; Zowghi et al. 2015). Moreover, when every end-user can provide feedback anytime and anywhere, several challenges for requirements elicitation arise, for example, processing of fake and very subjective feedback (Dalpiaz 2011; Johann and Maalej 2015).

2.2. Monitoring for Requirements Elicitation

Monitoring is commonly used to assess if the requirements are satisfied or violated during the execution of the system, i.e. requirements monitoring (Robinson 2009). Requirements monitoring comprises the analysis and evaluation of the stated requirements, tracing them to the metrics, observing and measuring the metrics of the system, and deriving the requirements status (Robinson 2009). Until recently, monitoring approaches have been focusing mainly on requirements assessment.

However, with the emergence of data-driven decision-making methodologies, new monitoring approaches have been proposed to support the requirements elicitation process. For instance, some approaches propose to use behavioural data to better understand end-user needs in web-based applications. For this, various types of data can be measured, including the end-users' action flow, their eye movements, and the length of time they spend on different features (Liu et al. 2017). In this direction, some approaches propose to specify the end-user goals accompanied with the expected end-user behaviour and then observe deviations on that behaviour to identify the need for a new requirement (Brill and Knauss 2011a).

Other approaches propose to use not only behavioural data but also runtime data of the system (e.g., QoS) to identify when a high-level indicator (e.g., reliability) is being violated. Such violation would, ultimately, lead to a new requirement (Guzmán et al. 2017).

2.3. Combining Feedback and Monitoring Data for Requirements Elicitation

A number of researchers have proposed to use both feedback and monitoring data to elicit new requirements. For instance, to support requirements elicitation, some approaches (Fotrousi et al. 2014) (Dzvonyar et al. 2016)combine feedback data with monitoring data coming from the same end-user who provided the feedback (e.g., log data). However, using this approach, the authors were only able to capture data of the end-user who provided feedback and were not able to capture data from other end-users (e.g., to identify how many end-users experienced an issue reported in feedback), or other types of monitoring data (e.g., QoS). In contrast, another approach (Dąbrowski et al. 2017) used monitoring data from all end-users and applied process mining techniques to observe end-users behaviour and elicit new requirements. In this work, the authors suggested that such information could be combined with feedback to refine the requirements and help improve the requirements prioritisation process. However, the research direction was not explored in-depth, and most of it was left as future work.

In fact, few technical solutions that combine feedback and monitoring data exist for other purposes. For instance, QoE probe (Fotrousi and Fricker 2016) is a lightweight mobile application that combines user feedback and monitoring data for requirements verification and validation. The probe periodically requests for user feedback, while collecting and aligning continuously with usage logs. The tool records usage data including user-id, timestamps of requirement events in the feature level (e.g., starting or completing a feature) and user interaction level (i.e., user input or application output). In completion of user interaction, a feature or a group of features, a feedback dialogue is triggered based on a defined likelihood, to collect the level of users' acceptance and user comments. The probe is only for gathering user feedback and monitoring data and does not include any analysis component.

MyExperience (Froehlich et al. 2007) is another tool that combines monitoring data and user feedback, but it is used to support studies on human behaviour or health (e.g., monitoring health-related metrics through sensors and asking end-users how they feel). However, to the best of our knowledge, no solution has advanced from a conceptual solution to a technical implemented framework that comprehensively combines feedback gathering and monitoring to support continuous requirements elicitation.

3. FAME Framework

We have developed a framework named FAME (Feedback Acquisition and Monitoring Enabler). FAME (available in https://github.com/supersede-project/monitor_feedback) enables a requirements engineer to collect and analyse the combined feedback and monitoring data regarding a software system. FAME was developed as part of the SUPERSEDE EU project.

Figure 7-1 presents a general overview of FAME. This overview shows schematically how FAME can support the requirements elicitation process. As shown, the *End-users* provide their *User feedback* through a *Feedback component*, whereas *Runtime events* from the *Host Application* are being monitored through a *Monitoring component* and stored in a *Data Lake*. The *Combiner* aggregates the data from both sources using an *Ontology* and presents the combined data to the *Requirements Engineer* who will elicit a *New Requirement*.

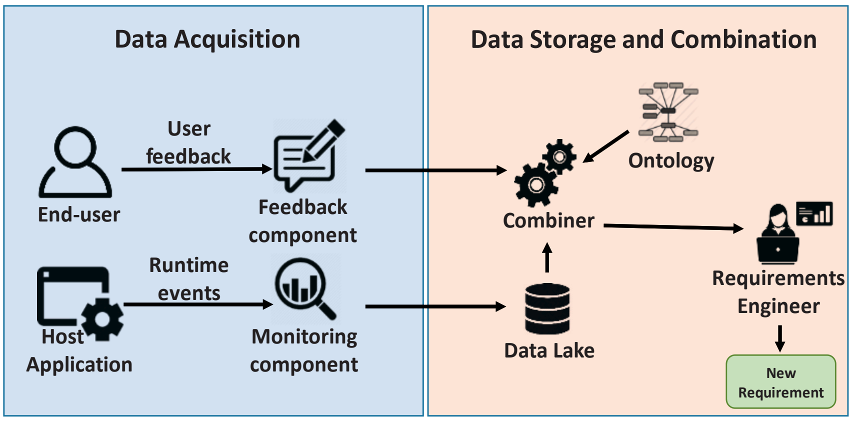


Figure 7-1. General overview of FAME supporting the requirements elicitation process.

A more detailed view of FAME is presented in Figure 7-2. The *Data Acquisition* components of FAME collect the user feedback and runtime events simultaneously (left side of Figure 7-2). The collected data is sent to the *Data Storage and Combination* components (right side of Figure 7-2), which store the data following a predefined structure. The data is then interpreted through an *Ontology* that integrates the semantics of both sources. In this sense, the requirements engineer has the combined information to assess the need for a new requirement and elaborate such new requirement.

3.1. Data Acquisition

The architecture of the *Data Acquisition* part of FAME is depicted on the left side of Figure 7-2. It is composed of three packages: (1) *Feedback Management Node,* (2) *Monitoring Management Node,* and (3) *Orchestrator Node*, where each package includes various components. User feedback is acquired using the *Feedback Management Node* package, which has been implemented for the Web and Android. The implementation is in the form of a library, which a developer integrates into a *Host Application*. For this, the developer inserts a few lines of code into the application code to invoke the functionalities of the user feedback acquisition. As an example for a web application, the piece of code includes a few links to stylesheets and scripts, as well as a hyperlink to the feedback button.

Feedback Management is the main component that manages and configures the feedback dialogue and finally sends the collected feedback to the *Data Storage and Combination* components. The feedback dialogue consists of one or several so-called *Feedback Mechanisms* including (i) a text feedback mechanism for free text comments; (ii) a rating feedback mechanism for classifying experience usage (e.g., star rating, emoticons); (iii) a screenshot feedback mechanism for visualising the feedback issue with a screenshot that can even be annotated with marks; (iv) an audio feedback mechanism for spoken feedback documentation; (v) a category feedback mechanism for feedback classification (e.g., problem, praise) by using multiple selection boxes, and (vi) an attachment feedback mechanism for additional file upload.

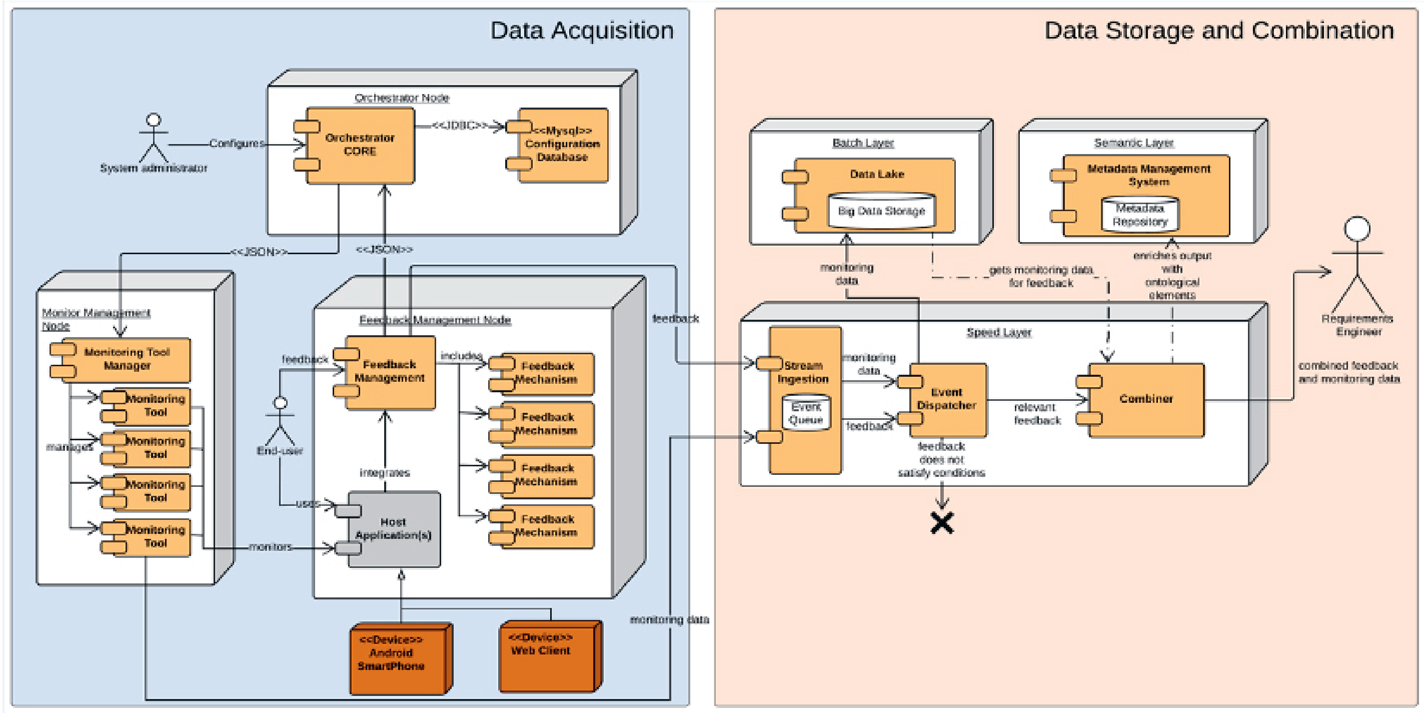


Figure 7-2. FAME architecture

Each time that a feedback dialogue is shown to an end-user, *Feedback Management* requests the latest configuration from the *Orchestrator Node* package, specifically the *Orchestrator CORE* component. The *Orchestrator CORE* component provides APIs that allow a system administrator to define or update configurations of feedback dialogues. The configuration received defines what feedback mechanisms, including what features and in which order they should be presented to the end-user. The configuration also defines how the feedback dialogue should be triggered either by the end-user, for example by pressing a feedback button (push), or automatically under certain conditions (pull). As soon as the *Feedback Management* receives feedback data from end-users, it transfers the feedback to the *Data Storage and Combination* components. The details are explained in Section 3.2.

Monitoring Data is collected through the *Monitor Management Node* package, which includes several *Monitoring Tools* following a Service-Oriented Architecture (SOA). The *Monitoring Tool Manager* is the main component of this package and manages the different *Monitoring Tools*. This component is a RESTful service that receives the instructions to run a particular monitoring task from *Orchestrator CORE* and dispatches the request to the specific *Monitoring Tool*, which can fulfil the required task. Examples of the *Monitoring Tools* include (i) a user events monitoring tool to obtain the clickstream and navigation path of end-users in web applications; (ii) an infrastructure monitoring tool to collect infrastructure related metrics (e.g., disk, memory, and CPU consumption of a server), and (iii) a QoS monitoring tool to compute the response time and availability of web services.

The system administrator can deploy *Monitoring Tools* of interest. After deployment, a *Monitoring Tool* may require additional integration activities. For the *Monitoring Tools*, the user events monitoring tool requires the integration of a JavaScript code in the *Host Application*. The infrastructure monitoring tool requires credentials to access the server to monitor through SSH. Only the QoS monitoring tool does not require additional steps, as it follows an active monitoring approach (i.e., it periodically invokes the web service and computes the response time and availability). The system administrator can then activate or deactivate integrated monitors, as well as (re)configure them on demand (e.g., change the monitoring frequency, the metrics to collect).

The system administrator can then activate or deactivate integrated monitors, as well as (re)configure them on demand (e.g., change the monitoring frequency, the metrics to collect).

If needed, a new *Monitoring Tool* can be implemented and integrated with the *Monitoring Tool Manager* (e.g., to carry out a monitoring task that is not supported by any of the *Monitoring Tools*). To do so, the system administrator should implement a RESTful service embedding the new *Monitoring Tool*. Such RESTful service should be compliant with an API (available in: https://github.com/supersede-project/monitor_feedback/blob/master/monitoring/src/main/jav a/monitoring/controller/ToolInterface.java) that is common for all *Monitoring Tools* and is used by the *Monitoring Tool Manager* to configure and run *Monitoring Tools*.

3.2. Data Storage and Combination

The *Data Acquisition* components of FAME can produce large quantities of data. In particular, the *Monitoring Tools* provide a continuous data stream consisting of runtime events from the *Host Application*. Moreover, end-users can provide feedback within the *Host Application*, which can also result in high-data volumes. As a result, FAME needs to deal with Big Data, which is supported by the *Data Storage and Combination* components of FAME (see right side of Figure 7-2). These components instantiate a Software Reference Architecture for semantic-aware Big Data systems (Nadal et al. 2017), which decouples Big Data processing into the *Speed Layer* (for real-time data processing) and the *Batch Layer* (for offline data processing). Its heart is the *Semantic Layer*, which enables data governance using Semantic Web technologies.

To process feedback and monitoring data from different sources, FAME exploits an ontology stored in the *Metadata Management System*. The ontology provides a formal, machine-readable conceptualisation of the domain of interest as well as the key concepts of feedback (e.g., *Rating*, *Message*) and monitoring data (e.g., *URL*, *ElementText*). Figure 7-3 depicts a fragment of the ontology at the high abstraction level as used by SEnerCon (see

Section 4), where feedback and monitoring data are linked via their shared schema elements (*User*, *Timestamp*, *Application*; bold framed, grey boxes in Figure 7-3). The feedback and monitoring data can also be related via one or more domain-specific concepts, which we generally depict as the *DomainConcept* element (bold framed box in the bottom centre of Figure 7-3). For instance, a feedback entry discusses an issue related to the domain concept invoice; thus we look for monitoring data also related to invoicing. The integration and mapping of different schema elements in the ontology are based on our approach for ontology-mediated queries (Nadal et al. 2019). We then detect that both feedback and monitoring data sources have shared schemas or domain-specific concept elements. This allows us to define a query that automatically joins both sources and presents an integrated result.

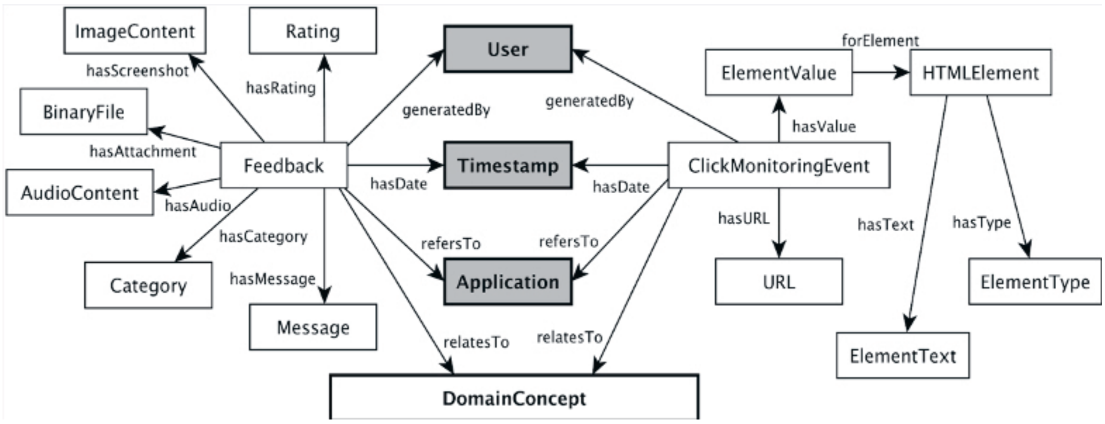


Figure 7-3. Excerpt of the ontology

In addition to data processing, FAME provides custom functions for data pre-processing, which can filter feedback relevant for requirements elicitation, such as feedback categorised as feature requests, or negative comments. To this end, FAME adopts machine learning techniques to automatically identify the feedback category (e.g., bug, feature request) and feedback sentiment (positive, negative, or neutral). We specifically employ Multinomial Naive Bayes classifiers for such predictive tasks (Guzman et al. 2017).

In the context of *Data Storage and Combination* components, data flows as follows. The *Stream Ingestion* receives data coming from *Feedback Management* and *Monitoring Tools*, which are then consumed by *Event Dispatcher*. The *Event Dispatcher* then checks and routes the data in the following way: Monitoring data is forwarded to and stored in the *Data Lake* - a massively distributed storage system - so that it can be retrieved for processing at a later stage. In parallel, the *Event Dispatcher* applies pre-processing techniques to the feedback data.

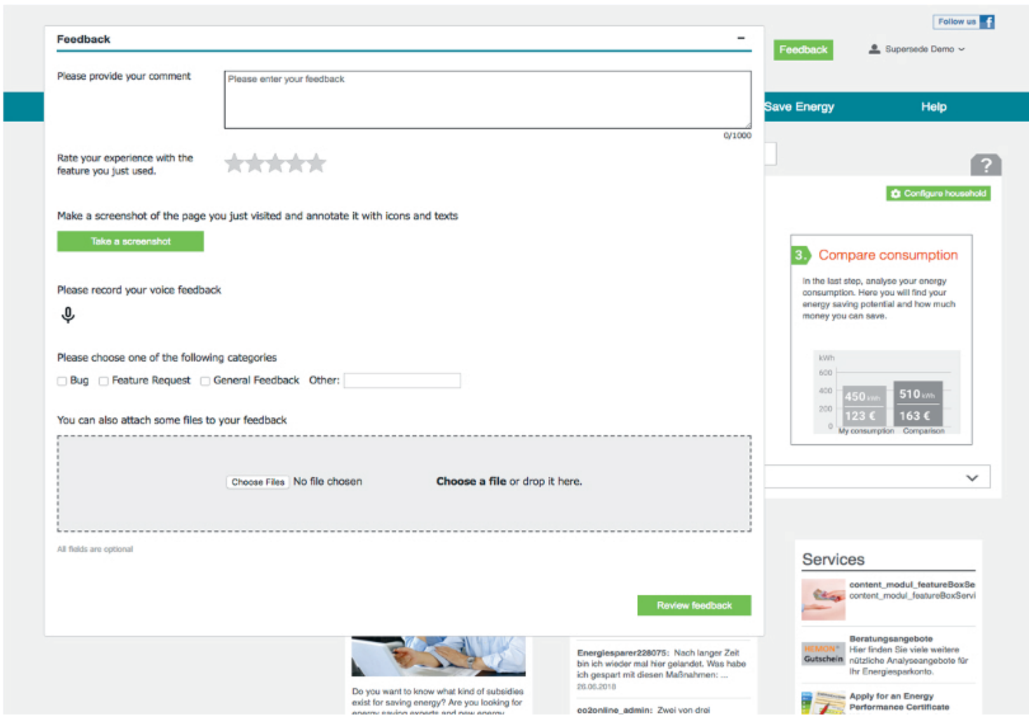
This allows the automatic filtering out feedback that may not be relevant for the requirements engineer. For instance, they might be interested only in feature requests with negative sentiment. Next, the output of the filtering function integrated into the *Event Dispatcher* is forwarded to the *Combiner*, which retrieves from the *Data Lake* the monitoring data that is related to the feedback data. The *Combiner* joins both types of data via their shared schema elements (i.e., User, Timestamp, and Application identifiers). Once combined, *Combiner* forwards the data to the requirements engineer for further interpretation.

It is worth noting that due to the flexible configuration features of FAME, several different options exist on how FAME can be implemented. Furthermore, the loosely coupled design of FAME enables the addition of more data sources and tools that the ones here presented.

4. Validation

4.1. Deployment and Configuration of FAME

We tailored FAME to the characteristics of the iESA (*Host Application*) and the needs of SEnerCon and SUPERSEDE. As a result, decisions about the feedback dialogue configuration received from the *Orchestrator Node* (e.g., mechanism order, instructions) were not only taken together with SEnerCon but were also driven by research aims. To obtain as much information as possible from the user feedback, we decided that the *Feedback Management Node* be deployed and configured to include all the *Feedback Mechanisms* available in FAME. As a consequence, all these feedback mechanisms are presented to the iESA end-user once they press a visible feedback button (push) that is available from every page in the iESA in Figure 7-4 (For the German feedback dialogue used in our validation, please see http://co2onl.in/5f21b50d). In this single-page feedback pop-up, the iESA end-user must first provide their comment in a mandatory text field with a limitation of 1000 characters (text feedback mechanism). Second, they can use a star rating to express their experience with the feature they used (rating feedback mechanism). Third, by using the screenshot feedback mechanism, they can take and edit (e.g., draw arrows) a screenshot of the page in the background. As fourth input, they can record their voice (audio feedback mechanism). Next, they can indicate the category (or categories) of their feedback, e.g. “Bug”, “Feature Request”, “General Feedback”, “Other”; category feedback mechanism). Note that for the feedback type classification we have decided to allow the feedback sender to choose multiple categories because we have seen in previous feedback that SEnerCon has received that their end-users sometimes communicate more than one issue in the same feedback message (e.g., starting with general praise, followed by a problem description).


Feedback

Please provide your comment: Please enter your feedback 0/1000

Rate your experience with the feature you just used. ☆☆☆☆☆

Make a screenshot of the page you just visited and annotate it with icons and texts

Please record your voice feedback

Please choose one of the following categories

☐ Bug ☐ Feature Request ☐ General Feedback Other:

You can also attach some files to your feedback

Choose Files No file chosen Choose a file or drop it here.

All fields are optional

3. Compare consumption

In the last step, analyse your energy consumption. Here you will find your energy saving potential and how much money you can save.

Energiesparer228075: Nach langer Zeit bin ich wieder mal hier gelandet. Was habe ich gespart mit diesen Maßnahmen: ...

Do you want to know what kind of subsidies exist for saving energy? Are you looking for

Beratungsangebote Hier finden Sie viele weitere nützliche Analyseangebote für Ihr Energiesparkonto.

Figure 7-4. Feedback dialogue

As the last input, the iESA end-user can upload further files (attachment feedback mechanism). After providing all their input, they click on the “next” button, check the summary of their feedback in a second dialogue page, and finally press the “send” button.

The *Monitoring Management Node* was deployed and configured with just one *Monitoring Tool*. In particular, SEnerCon was interested in monitoring the behaviour of its end-users, and hence, we deployed the user events monitoring tool to obtain the clickstream and navigation path of end-users in iESA. The infrastructure and QoS monitors were not deployed since SEnerCon was not interested in the metrics gathered by these *Monitoring Tools*. In the Terms of Service, which was included in the registration form end-users accept before using the iESA web application, end-users were informed they were being monitored.

Data Storage and Combination components were deployed without configuring any filter for the feedback in the *Event Dispatcher*. This is because the amount of feedback expected by SEnerCon about iESA, based on their experience with their other feedback communication channels, was not large enough to justify a filter and SEnerCon wanted to analyse all the feedback obtained without any pre-processing. Nevertheless, filters might be required in other cases, such as when there is a significant amount of feedback received that is not relevant for the generation of a new requirement.

4.2. Validation Protocol and Execution

The validation protocol distinguished two key stages: feedback and monitoring data gathering; and requirements elicitation using such data.

The feedback and monitoring data were collected between October 1st, 2017 until January 31st, 2018 (4 months). The requirements elicitation process was conducted through a workshop. The workshop involved a researcher and an employee from SEnerCon (both are authors of this paper). The employee who acted as SEnerCon representative was one of the leading developers in the production of the iESA.

The workshop was divided into two phases. In the first phase, the SEnerCon representative had to elicit requirements considering only feedback data (as SEnerCon has done so far). In the second phase, they had to elicit further requirements or refine the previously elicited ones considering the combination of feedback and monitoring data. With this validation procedure, we were able to assess the benefits of combining feedback and monitoring data concerning using just feedback data as an information source in the requirements elicitation process.

To execute the first phase of the workshop, the researcher presented the feedback obtained by FAME to the company representative who identified which feedback entries were relevant to elicit a requirement (as not all feedback might lead to a requirement). Then, for the feedback entries that were identified as relevant, they elicited the requirement and documented it by filling in a predefined template (see Table 7.1). This template included the following fields: the id of the requirement; the ID of the feedback entry that triggered the requirement; a description of the requirement (with an optional satisfaction criteria); the priority of the requirement (with five possible values from very high to very low); and a field to document other observations that the representative may consider relevant. An example of a requirement obtained in the first workshop phase is also depicted in Table 7.1. The execution of the first workshop phase lasted two hours and fifteen minutes with thirty minutes to identify which feedback entry was relevant, and one hour and forty-five minutes to elicit and document the requirements.

To execute the second workshop phase, the researcher provided the relevant feedback entries, identified in the previous workshop phase, combined with the monitoring data to the company representative. The presented monitoring data included the clickstream and navigation path of the end-user who provided the feedback and how many end-users used the same functionality. While going through the combined feedback and monitoring data, the company representative identified further requirements and modified (some of) the previously documented requirements. The researcher annotated these changes. The execution of the second workshop phase lasted two hours and thirty minutes.

Table 7-1. Example of an elicited requirement after the first workshop phase

Requirement ID	R1
Feedback ID related to the Requirement	F3
Description of the Requirement	The user should be able to see meter readings of all meters, not depending on the state of the meter (exchanged, active, smartmeter) in the Android application.
Priority of the Requirement	Low
Other Observations (if any)	None

4.3. Validation Results

FAME was successfully used in practice. In the defined period, FAME collected thirty-one feedback entries from twenty-four end-users. Figure 7-5 summarises how the end-users used the feedback mechanisms to communicate their feedback.

End-users categorised the feedback as “Bug”, “Feature Request”, “General Feedback” (multiple categories were allowed per each feedback); the category “Other” was not chosen. Screenshots and attachments were used in seven cases, and star-ratings were provided in twenty-four cases.

In the same period, FAME collected user events from all 5.185 end-users who logged-in during the period, regardless if they provided feedback or not. FAME collected 957.260 clicks in the clickstream (including 936.740 left-clicks, 6.547 right-clicks, and 13.973 double-clicks), and 160.888 navigation actions (i.e., steps of the navigation path). As an example, an excerpt of the clickstream of one end-user is shown in Fig. 6 (translated from German).

In the first workshop phase, sixteen out of thirty-one feedback entries collected with FAME were identified as relevant by the company representative and led to nineteen requirements (in three cases, the feedback entry led to two requirements). The fifteen feedback entries that were considered not relevant to elicit a requirement were bug reports and customer service related issues.

In the second workshop phase, the company representative analysed the combined feedback and monitoring data. The combined monitoring data covers the time span between user login and when the feedback is sent and includes the actions of the end-user who provided the feedback, as well as the list of end-users, who did not provide feedback but used the same actions as the feedback provider. It is worth noting that by applying the ontology-mediated queries, only the monitoring data related to the sixteen relevant feedback points were considered. From 957.260 clicks, only 2.164 were presented and analysed by the SEnerCon representative.

During the analysis of the combined data, the company representative found one additional requirement and modified four requirements they had documented in the first workshop phase. Below we describe some examples of how those requirements were modified as well as how the additional requirement was identified.

Monitoring data proved to be useful to either confirm or refute the priority of a requirement. For example, an end-user requested that an existing feature in the iESA web application should also be present in the Android application. Once the feedback was analysed, the requirement was considered "low priority" as the feature was not very relevant. Monitoring data confirmed such perceptions, as this feature was only used by 11 out of the 5.185 iESA end-users in the web application during this period.

In another example, an end-user requested a new feature to transfer data from their old household to a new household. One feedback from another end-user seemed to be about something very similar. As a first impression, such feature request seemed very relevant, and the documented requirement was assigned with a "high priority". Monitoring data, however, disproved such prioritisation as the number of end-users who looked for the data of their old household and had a new household in this period was just two, including the end-user who provided the first feedback in this example. For the second similar feedback, the monitoring data showed what the end-user was trying to achieve and clarified that they were asking for something else; they wanted to enter data from previous years to the same household instead of transferring old data to a new household. Ultimately, this led to an entirely different and new requirement.

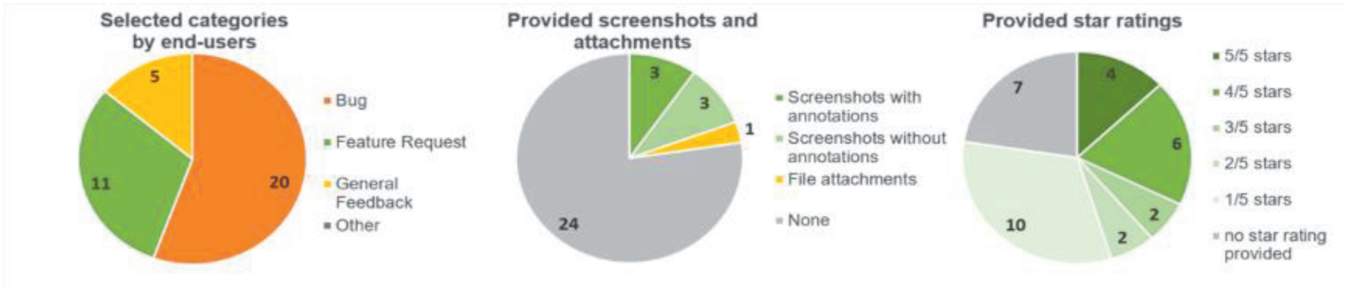


Figure 7-5. Overview of feedback collected with FAME

timestamp	event Type	element type	element ID	text	element value	URL
01/10/2017 4:36:03	click	INPUT	meter reading			https://www.energiesparkonto.de/esk/content/startPage/?portal_id=co2online
01/10/2017 4:36:09	click	INPUT	enterMeterreading_save		Save meter reading	https://www.energiesparkonto.de/esk/content/startPage/?portal_id=co2online
01/10/2017 4:37:30	click	SELECT	view	Years \t Months \t Weeks \t Days\t Hours \t	year	https://www.energiesparkonto.de/esk/content/bereichePage/?bereich=strom
01/10/2017 4:37:32	click	OPTION		Months	month	https://www.energiesparkonto.de/esk/content/bereichePage/?bereich=strom
01/10/2017 4:38:21	click	SELECT	view	Years \t Months \t Weeks \t Days\t Hours \t	month	https://www.energiesparkonto.de/esk/content/bereichePage/?bereich=strom
01/10/2017 4:38:23	click	OPTION		Days	day	https://www.energiesparkonto.de/esk/content/bereichePage/?bereich=strom
01/10/2017 4:39:07	click	A	logout	Logout	undefined	https://www.energiesparkonto.de/esk/content/bereichePage/?bereich=strom

Figure 7-6. Excerpt of the clickstream of one end-user collected with FAME

(columns element ID, text and element value have been translated here from German to English).

Monitoring data was also used to decide among two possible variants of the same requirement. For example, an end-user requested that the system should describe what “conversion factor” meant and how it should be calculated when entering consumption data taken from the gas bill. Considering only the feedback entry, a first conclusion by the SEnerCon representative was that the iESA web application should compute this “conversion factor” automatically when the end-user enters other parameters of the gas service. However, monitoring data of this end-user showed that they did not struggle to understand and compute the needed “conversion factor” as they spent less than one minute to introduce the “conversion factor” and all the data from the gas bill. Moreover, such functionality was only used by 203 end-users and in total 332 times. This means that end-users who use that functionality use it on average less than twice in a four-month period. With this information, the effort to implement a functionality to compute the “conversion factor” was not justified, and, the requirement was documented as “the term conversion factor should be explained when requested in the gas consumption form”.

4.4. Threats to Validity

Threats to Internal Validity. The first threat is a possible bias in the data analysis conducted by the company representative. SEnerCon is a partner in the SUPERSEDE research project and is aware of the effort the research partners spent on the development of FAME. As a result, we encouraged the representative to share any thoughts with us, including critiques or doubts. Moreover, the validation workshop was moderated by the main author, and they might have unconsciously guided the requirements elicitation process in the research authors’ desired direction. To limit the workshop moderator’s influence, the research authors had defined a protocol before the validation execution including a semi-structured workshop guideline. The workshop moderator was also not allowed to express their own opinions.

A second threat is about the requirements elicitation expertise of the company representative. Although they are an expert in the iESA web application and knows the existing requirements, they are not an expert in requirements elicitation. In the workshop, this might have influenced the number of feedback points they indicated as relevant for requirements elicitation, the final number of elicited requirements as well as their modifications.

Third, the low number of feedback points obtained is limiting our evaluation method. We have been able to prove the feasibility of FAME and provided an initial qualitative evaluation that demonstrated how FAME improves the requirements elicitation process. However, we could not conduct a quantitative evaluation as it would require a higher amount of feedback data points collected with FAME.

Threats to External Validity. One limitation to the generalizability of our results is that only one member of one software company was involved in our validation study. Secondly, regarding the selected *Host Application*, we have focused on the integration of FAME in a web application in the domain of energy saving. Thirdly, we run our feedback and monitoring data collection with only one particular configuration of FAME. Finally, we could not investigate the support of FAME for requirements engineers with different expertise levels.

5. Conclusion and Future Work

In this paper, we have presented FAME, a framework for the combined and simultaneous collection of feedback and monitoring data in Web and mobile contexts to support continuous requirements elicitation.

FAME proposes managing both acquired feedback and monitoring data through the same infrastructure as well as combining and structuring them employing an ontology. The FAME architecture is split into two main groups of components: *Data Acquisition* components, and *Data Storage and Combination* components. The *Data Acquisition* components integrate a feedback acquisition tool with multiple *Feedback Mechanisms* and a monitor manager with multiple *Monitoring Tools* that collect user feedback and runtime events respectively. The *Data Storage and Combination* components instantiate a Software Reference Architecture for semantic-aware Big Data systems that structure and combine the collected data via an ontology that maps the relationship between user feedback and runtime events.

To validate our solution and assess in which situations the combination of monitoring and feedback data provided valuable information to elicit new requirements, we deployed FAME in a web application of a German SME and conducted a requirements elicitation process through a workshop using the data obtained by FAME. Results showed that FAME was deployed and used in an industrial context to combine feedback and monitoring data, also bridging a research gap in RE. Moreover, our first validation results identified a few examples where the combination of feedback and monitoring data could improve the requirements elicitation process in contrast to just considering only the feedback. Due to the small data set and the data fragments used in our study, conclusions regarding quantitative, situation-specific benefits of combined feedback and monitoring data cannot be drawn. However, we assess our presented flexible and extensible FAME framework as a robust tool solution to advance research on combined feedback gathering and monitoring as a mean to support continuous requirements elicitation.

As future work, we plan to validate FAME with companies of different domains, sizes, and requirements elicitation processes, as well as with different amounts of end-users involved. This future validation will be on various configurations of FAME including both web and mobile *Host Applications*, more *Monitoring Tools*, and various filters for data pre-processing. Such future work will allow us to assess the benefits of combining user feedback and monitoring data and to derive recommendations and best practices, e.g., which amount of which data types in which aggregation level are most useful to combine feedback and monitoring data.

We also plan to extend FAME in several directions. For example, we may use the monitoring data first to trigger a new requirement and then combine the monitoring data with the related feedback. Moreover, we may integrate further user feedback channels, such as social media into the FAME framework; and further *Monitoring Tools*, such as sensors embedded in mobile devices. Because FAME requires a continuous flow of feedback data to support continuous requirements elicitation, we also plan to investigate how we can best encourage end-users to provide feedback. This, in turn, might result in further extensions of FAME, for example, feedback-to-feedback or gamification components. Regarding the *Data Storage and Combination* components, we plan to incorporate automatic clustering of feedback data and advanced aggregation of monitoring data. This would improve how the information is presented to the requirements engineer to elicit requirements (e.g., grouping similar feedback, showing monitoring data with computed metrics instead of only runtime events).

Acknowledgment

The authors thank the anonymous reviewers. A special thanks goes to Colin Venters for his support. This work was supported by the European Commission within the SUPERSEDE project (Agreement No. 644018).

Chapter 8 : A Method for Gathering Evidence from Software-in-Use to Support Software Evolution

[The chapter is based on: ***F. Fotrousi****, S. Fricker, M. Fiedler, D. Wüest (2020). "A Method for Gathering Evidence from Software-in-Use to Support Software Evolution", Submitted to a Journal.*]

Abstract

Companies need to acquire evidence about situations where their running software product behaves inappropriately, which users often do not accept the software. Gathering and combining system monitoring with user feedback allows the company to acquire knowledge about the software product and find evidence that can support decisions for software evolution. So far, monitoring data and feedback have been collected passively, hoping that users become active when problems emerge. This approach leaves unexplored opportunities for software product evolution when massive amounts of monitoring data is collected and users do not provide feedback for matters of interest to the product development team, which we simply call product team in this article. Furthermore, the research literature does not provide a clear understanding of the conceptual process behind the gathering and organization of such knowledge. In this paper, using a design science research method, we propose a method for the gathering of evidence from software-in-use (GESU). The GESU method is designed within the frame of knowledge creation theory. We conceptualize the method and frame its processes using theoretical support. The method also addresses some technical issues by combining goal-based system monitoring with proactive, autonomous user feedback collection and surface knowledge of the system use that is relevant for system maintenance and evolution. It monitors product goals to identify interesting situations of system use and issues automated requests for user feedback to interpret the product impact from the user's perspective. The GESU was evaluated in a smart city case. The evaluation results showed that the GESU could effectively support decisions for software evolution in the case. The finding confirmed that the GESU could describe the knowledge creation theory presented using the SECI (socialization, externalisation, combination and internalization) model, although it needed some adaptation to the case. As further contributions, the paper revisited the SECI model as the conceptual basis of gathering knowledge from software-in-use in the context of system maintenance and evolution.

Keywords

Gathering Evidence, Knowledge Creation, System Monitoring,
User Feedback, Software Evolution

1. Introduction

Software maintenance and evolution constitute a large part of the work of software companies (Kittlaus and Fricker 2017). Professionals in these companies study the use of the software system and user feedback to identify user needs, software bugs and fixes that can be translated into requirements and software improvements (Olsson et al. 2012; Stade et al. 2017) for future software evolution (Lehman and Ramil 2003). The value of such studies is to identify evidence for evolution and validate the developers' assumptions about user needs and system issues when the system behaves inappropriately, which is not desirable or acceptable for the end-users or other stakeholders.

One of the challenge of software evolution is to identify and integrate evidence from a variety of sources such as bug reports, error logs, user reports, software processes and its architecture in order to study the product and determine required changes (Lehman and Ramil 2003).

A considerable amount of evidence for software evolution (which will simply be referred to as evidence in this article) can be gathered from software-in-use. Diverse methods and tools have been developed to study evidence, for example, by monitoring the product's use (Chapin et al. 2001), gathering user feedback (Maalej et al. 2016), and checking whether a product meets the a user's desires and needs (Fickas and Feather 1995). The systems are monitored at runtime (Fotrousi and Fricker 2016). The feedback is commonly gathered with hotlines, email, contact forms, ticket systems, feedback forms embedded in the software and with user feedback mechanisms in app stores (Stade et al. 2017). This data has been generated in the early stages of the development lifecycle when a product or its features are tested with prototypes (Ali et al. 2011) and in later stages when it undergoes evolutionary change (Bosch 2012) or is being maintained (Carreño and Winbladh 2013).

Approaches have also been developed to mine and analyse the data with the aim of extracting requirements (Guzman and Maalej 2014) and supporting decisions for system evolution (Kittlaus and Fricker 2017). A systematic method for such decision-making about system evolution is the innovation experimentation method (Blank 2013; Edison et al. 2018). With innovation experiments, the organisation systematically prototypes ideas about changed product capabilities and collects feedback about their impact to understand whether the ideas are promising and should be fully developed and integrated into the product that is deployed in the market, or whether they should be abandoned.

While each piece of gathered data may be useful in and of itself, it is in its combination that it is expected to offer the organization actionable knowledge of system use (Oriol et al. 2018). Isolated monitoring data may turn out to be difficult to interpret and irrelevant; also, badly timed requests for user feedback risk disturbing users and may result in feedback disconnected from the context to which it applies (Fotrousi et al. 2018). So far, to the best of our knowledge, no previous work (even that of (Oriol et al. 2018) provides a theoretical foundation and process description for how to approach the gathering and combination of evidence to offer support for decisions about system evolution.

This article aims at addressing this gap by introducing a knowledge gathering and combining method and validating its practical use in a case study of product prototyping. The work is situated in a larger undertaking that follows design sciences research (Peffers et al. 2007) and is shown as one iteration of implementing the method of "gathering evidence from software-in-use" (GESU) underlying the processes for a real-world product. The method has been developed based on Nonaka's theory of creating knowledge (Nonaka and Takeuchi 1991) and adapted for the gathering, sharing and aggregating of evidence that can be used for decision-making about product evolution. Our method has been validated with an interpretive case study of an organization that developed a smart city application for city visitors. We collected and combined knowledge from system and usage measurements as well as user feedback. The knowledge was translated into evidence for software evolution to understand what and why to evolve. We then interviewed four members of the product development team related to the case to validate how they coped with the knowledge in order to validate our proposed model. The results showed how the method may be applied in a real-world software organization and how the organization may structure the collected data and work process to exploit the benefits of the method.

This article extends our earlier paper on combining monitoring and autonomous feedback requests to gather actionable knowledge of system use (Wüest et al. 2019). We extended the paper by introducing the GESU method derived from Nonaka's model of knowledge creation (Nonaka and Toyama 2003) and further described the smart city case study for a more comprehensive validation of the method.

The remainder of the article is structured as follows. Section 2 introduces the knowledge creation model and describes how it is related to the theoretical background. Section 3 discusses the research problems, and Section 4 proposes the GESU to solve them. Section 5 explains the research methodology. Sections 6 and 7 describes the application of the method to a smart city case study and its evaluation, respectively. Section 8 discusses the obtained results and future work. Section 9 summarizes and concludes this article.

2. Gathering and Sharing Evidence: Background

The evolution of software is done in response to requests for new features, the existence of new platforms and the desire to improve the software quality, while considering preventing factors including market saturation, political and legal concerns and software complexity (Godfrey and German 2008). The evolution is based on evidence that software practitioners have gathered, plus their personal experience (Devanbu et al. 2016). The gathering of evidence is usually performed through observation and measurements in laboratory experiments, real cases or a literature review of previous practices (Kitchenham et al. 2015).

In recent years, a new software evolution practice has been seen; the lean start-up approach uses experimentation to achieve a product-market fit (Blank 2013). Companies that embrace the lean start-up approach follow a highly iterative process of product development in which many opportunities are created to gather feedback for product testing, thus obtaining evidence for evolving the product concept. Even though the lean start-up movement is just a few years old, it has gained rapid acceptance in industry, both for start-ups as well as established companies (Edison et al. 2018).

The lean start-up experiments are centred on hypotheses of value creation with a product that is under development (Eisenmann et al. 2012). These hypotheses capture a capability of the product and the impact that this capability is expected to create on the environment in which the product will be used (Osterwalder et al. 2014). The process requires developing a vision, building a prototype (the "minimum viable product"), observing the user reactions, learning from the observations and deciding on whether to pivot by repeating the process with a new hypothesis. Once enough support is found for a hypothesis, the product concept is used for the development of the product and scaling the new product version in the market.

While a hypothesis-driven approach is favoured in the lean start-up approach (Eisenmann et al. 2012), other ways of creating and sharing evidence from a product may be pursued as well. For example, in requirements elicitation, a rich body of knowledge exists on how to gather evidence from end-users and stakeholders and communicate them to the developers of a software product (Carrizo et al. 2014; Sutcliffe and Sawyer 2013). Still missing is an understanding of the essence of gathering and sharing product evidence. This section develops the theoretical background and suggests a way of operationalising the theoretical viewpoint in software engineering.

2.1. A Theory for Gathering and Sharing of Evidence

In general, evidence is "the available body of facts or information indicating whether a belief or proposition is true or valid" (Oxford 2019). Evidence underpins knowledge, and we expect that knowledge is derived from evidence through some process of interpretation (Kitchenham et al. 2015). The literature and theories around the definition, gathering and sharing of knowledge that inspired us in this study were fairly mature.

According to the theory of knowledge, or epistemology, knowledge is defined as "justified true belief." In other words, knowledge is a belief connected with a fact known in the right way. Although the belief could be made based on knowledge acquired from external sources, personal experience plays an important role in forming personal knowledge (Devanbu et al. 2016). In our study, we consider knowledge as a "correct information sense" (Lehrer 2018) to highlight the essential role of knowledge in human reasoning about what is true and what is false.

Knowledge is either tacit or explicit (Collins 2010): tacit knowledge exists in the mind of the human actors, and explicit knowledge is documented. Tacit knowledge is highly personal and difficult to articulate. The knowledge exists in the forms of subjective insight and intuitions and consists of mental models and perspectives. Explicit knowledge is expressible and readily sharable with others in the form of documents, manuals and specifications.

Nonaka and Takeuchi (1991) have defined a theory of knowledge creation. The theory focuses on knowledge creation as a process of making tacit knowledge explicit. It indicates that knowledge is refined by passing through different modes of conversion from tacit to explicit knowledge and vice versa. Nonaka and Toyama (2015) later revisited their theory of knowledge and updated definitions in considering the theory as a synthesizing process. Figure 8-1 shows the updated theory of the knowledge creation model. The model defines the four knowledge processes (socialization, externalization, combination and internalization, or SECI) as follows:

- *Socialization* is sharing and creating tacit knowledge through direct experience, observation, imitation and practice.
- *Externalization* is articulating tacit knowledge through dialogue and reflection to make it explicit. The explicit knowledge can be in the form of metaphors, analogies, concepts, hypotheses or even models.
- *Combination* is taking explicit knowledge from different sources such as documents, user feedback, videos etc. and aggregating and systematizing it.
- *Internalization* is taking the combined knowledge and turning it into individual tacit knowledge in the form of mental models or technical know-how.

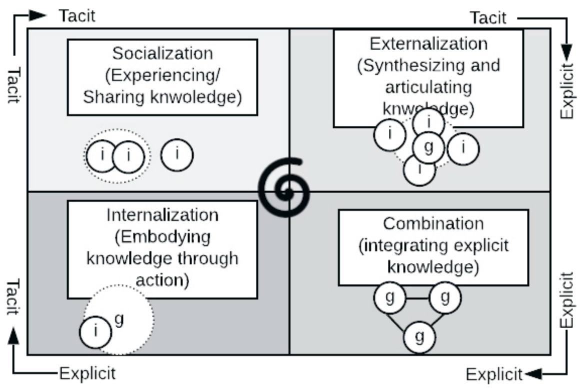

Tacit
Tacit
Tacit
Explicit
Socialization (Experiencing/ Sharing knwoledge)
Externalization (Synthesizing and articulating knwoledge)
Internalization (Embodying knowledge through action)
Combination (integrating explicit knowledge)
i
g
Tacit
Explicit
Explicit
Explicit

Figure 8-1. The knowledge creation SECI model (Nonaka and Toyama 2003)

Knowledge is created between individuals, individuals and the environment or among a group of individuals, and the SECI model describes how organizations create and share the knowledge. In Figure 8-1, individuals (*i*) and groups (*g*) are represented by circles marked with individual letters. Nonaka and Takeuchi (1991) argued that individuals initially create knowledge, and the knowledge becomes organizational knowledge through the knowledge creation process described by the theory. In *socialization*, individuals experience knowledge or share their knowledge through actions like face-to-face sharing. In *externalization*, individuals belonging to a group discuss a concept to come to a shared understanding and document it, while in *combination*, the explicit knowledge from groups is integrated. An individual learns the explicit knowledge by performing some action through *internalization*.

Software engineering is knowledge-intensive work and several studies have considered the development of knowledge, management of knowledge and the use of knowledge (Bjørnson and Dingsøyr 2008). There are also studies that have adapted the SECI model to fit the context of software engineering. Hansen and Kautz (2004) used the model for identifying knowledge flows in a software organization. The study replaced the externalization process of the SECI model with *codification strategy*. Bider and Jalali (2016) took the SECI model for traditional business process development where *adoption* and *embedment* replaced socialization and internalization, respectively. They then adapted the knowledge transformation into agile process development by removing the combination process.

To acquire evidence for software evolution, we selected to use the SECI model for framing the gathering. However, none of the previous adaptations of SECI in software engineering were appropriate to the context of this paper. Therefore, the original version of the SECI model was utilized in this study. In addition to the theoretical framework, it was also important to understand previous works regarding the technical dimension for gathering knowledge. These are described in the next section.

2.2. Methods for Gathering Evidence

There are various methods and tools to obtain knowledge about software products. Workshops and meetings are examples of approaches for gathering knowledge from participants. This knowledge can contribute to identifying evidence for evolving software systems. This section gives an overview of the approaches for gathering knowledge and evidence from a system at runtime or from a user.

2.2.1. Gathering Evidence from a Running System

Monitoring a system at runtime allows engineers to determine whether or not and to what degree the implemented system is meeting the requirements of its users (Carreño and Winbladh 2013). The insertion of code or sensors into a running system allows the developers to continuously check the system's health, observe the users, record their activities and study the system's behaviour (Wellsandt et al. 2014). Such monitoring enables engineers to monitor goals, detect requirements violations (e.g., system failures) and react quickly to evolve the system (Leucker and Schallhart 2009). Furthermore, observing user activities, such as the sequence of features used, duration and other contexts, enables requirements engineers to understand user needs better (Maalej et al. 2016).

Several approaches have been studied for monitoring a system and its requirements at runtime (Rabiser et al. 2017; Vierhauser et al. 2016). Some rely on continuous observation of the system use and continuously analyse the recorded logs to identify evidence for changes (Fotrousi and Fricker 2016). There are other studies in which the system use is monitored based on a requirements or goal model (Goldsby et al. 2008; Wang et al. 2009). The benefit of the second approach is that the data gathering is more focused and a lighter analysis is introduced to find evidence for changes when compared to the first approach. (Qian et al. 2018) proposed a goal-driven framework that monitors the adherence of user behaviour to the goals. It enables adaptive activation of the goals based on the behavioural information to reason about the fulfilment of the goals.

2.2.2. Gathering Evidence from Users

There are several approaches to gather user knowledge and therefore evidence for software evolution. Workshops for the gathering of requirements (Fricker et al. 2015), online games to gather user preferences (Hacker and Von Ahn 2009) and specific product reviews in an app store (Kurtanović and Maalej 2018) are examples of such approaches.

Feedback given by users is another source of information to understand user needs and how satisfied the users are with the system (Knauss et al. 2009). Several feedback tools and approaches have been designed to collect such information with user feedback. Feedback tools are either offered as a standalone option or are embedded into the system (Fotrousi and Fricker 2016; Seyff et al.

2014). The feedback tools trigger feedback forms either by a user's request, such as pressing a feedback button, or by a system request, such as with an automatic pop-up window (Maalej et al. 2009). Feedback forms enable users to communicate bug reports, feature requests and praise (Maalej and Nabil 2015). The feedback may be collected as a simple combination of free text, selected categories, ratings and screenshots with annotations (Elling et al. 2012; Morales-Ramirez et al. 2015). Regardless of the dialogue design, several studies describe challenges of analysing and interpreting user feedback, especially when information about the context in which it applies is missing (Pagano and Brügge 2013; Stade et al. 2017).

2.2.3. Combining Evidence

Combining monitoring data and user feedback can create new knowledge while mapping the feedback within the context in which it applies (Oriol et al. 2018). There are a few studies that have contributed to creating and using such combined knowledge. Seyff et al. (2014) proposed to connect user feedback with features of the user interface to improve usability. Fotrousi et al. (2014) proposed to correlate the users' quality of experience with the system's quality of service to elicit non-functional requirements. Oriol et al. (2018) proposed a generic framework for combining the collection of feedback and monitoring data and tested the framework for requirements elicitation. Furthermore, Mattos et al. (2018) presented an experimentation activity model where it was confirmed that user feedback along with monitoring data could contribute to an engineer's knowledge.

3. Research Problem

The previous studies discussed confirm that the combination of monitoring data and user feedback contributes to creating new knowledge that could enhance the knowledge of a product development team. A product development team could not achieve some of this evidence with either monitoring data or user feedback alone. Prototypes have been developed to show the feasibility of gathering such knowledge in practice (Oriol et al. 2018; Seyff et al. 2014). However, the prototypes had some drawbacks.

First, the collected user feedback lacked sufficient information about the context in which the feedback was given. Oriol et al. (2018) only validated the solution with passively collected user feedback. An analysis of the passive user feedback could not describe the context if the user did not mention it in the feedback. Such a passive approach limits developers in targeting feedback collection on interesting situations of system usage and increases the risk of collecting irrelevant or even fake feedback (Dalpiaz 2011).

Second, the monitored data turned out to be challenging to analyse and interpret in continuous monitoring, especially when the monitoring of the software use for years created a massive amount of data. Such large data sets with an amount of data interconnection may require complex analysis in order to extract some knowledge about the product (Jin et al. 2015).

Third, the relevant studies have not provided a clear understanding of the conceptual process behind gathering and organizing evidence for evolving the software. The literature does not present how the evidence integrates into the stakeholders' knowledge and enhances it for deciding about the next release of the system.

To summarize, the *study problem* is to describe how software practitioners gather evidence from the use of a software system to support decisions about evolving the system, while ensuring the usefulness of the data gathered.

This paper proposes a solution for the *gathering of evidence from software-in-use* (GESU) considering two perspectives. From one perspective, we will conceptualize the method and frame its processes using theoretical support. From another, we will address technical issues and identify opportunities to evolve the gathering of evidence from software-in-use.

4. GESU: A Method for the Gathering of Evidence from Software-in-Use

As discussed earlier, monitoring data and user feedback are two sources of evidence used for evolving a system. The issues discussed in Section 3 engaged us to extend the feedback acquisition and monitoring enabler (FAME) approach (Oriol et al. 2018) by combining system monitoring and user feedback with autonomously generated proactive requests for user feedback.

We used the conceptual framework of SECI to justify the components needed for the method to guide data collection and to analyse the research. At first glance, none of the previous adaptations of the SECI model for software engineering (Bider and Jalali 2016; Hansen and Kautz 2004) worked for our concept, while the main SECI model fit it the best. The framework allowed us to explain the knowledge creation phenomenon while adopting the other researchers' points of view on this concept.

The SECI model was limited though. SECI only focuses on "human" individuals or groups and does not take the "machine" into account. However, the software product running on a machine is an important source of knowledge. We believe there is no critical difference between transferring tacit knowledge of an individual to another individual and transferring knowledge embedded in the software to an individual, or even to another piece of software.

Figure 8-2 presents GESU method in two dimensions: the knowledge transformation process and activities involved in the process. The model presents four processes of knowledge transformation (i.e., experiencing the software product, articulating knowledge, combining knowledge and operationalizing the knowledge) that correspond one-to-one with the processes defined in the SECI model (i.e., socialization, internalization, combination and externalization). For clarity, we adapted the terminology that the original SECI model uses. The method contains the seven activities explained below.

1- Preparing and running a software system. A company launches a software product or software system with a particular set of goals and lets its users, including end-users and stakeholders of the product, experience it. This is the first activity in transferring knowledge that exists within a software product (i.e., "the machine") to users (i.e., "a human"). For example, when a software feature is broken, the user knows the problem as he or she experiences the feature for the first time.

2- Monitoring the system and its usage. The goal-driven product monitoring occurs continuously behind the scenes. The occurrence of an internal event, such as a deviation in a measurement from its accepted threshold value, triggers an action. The accepted threshold is defined based on product goals. Using event-driven monitoring allows the product team to focus on those measurements that create an action for business growth. Tracking every single user click might not lead to generating actionable knowledge for the product team, while candidate measurements aligned with the product goal might. Goal-driven monitoring does not mean that one should collect only particular measures; it means that one should rely on the knowledge extracted from candidate measures. The other measures may help to reveal or explain a situation in the future.

3- Collecting user feedback. During this activity, feedback from users about the running product is collected. The method suggests proactive, autonomous requests for user feedback (Wüest et al. 2019) when an interesting situation in the use of a system is detected. Proactive requests for user feedback mean directly asking for feedback as through a pop-up feedback form. Autonomous user feedback means to customize the feedback form based on the observation of the system.

When the monitoring system (i.e., goal-driven monitoring) detects a deviation in a measurement from the accepted threshold (defined based on the goals), the system triggers a request for user feedback. In response to the triggered request, users share their perceptions, experiences or needs via user feedback that may explain the measurement deviation. The users themselves may also trigger the feedback form and provide similar feedback.

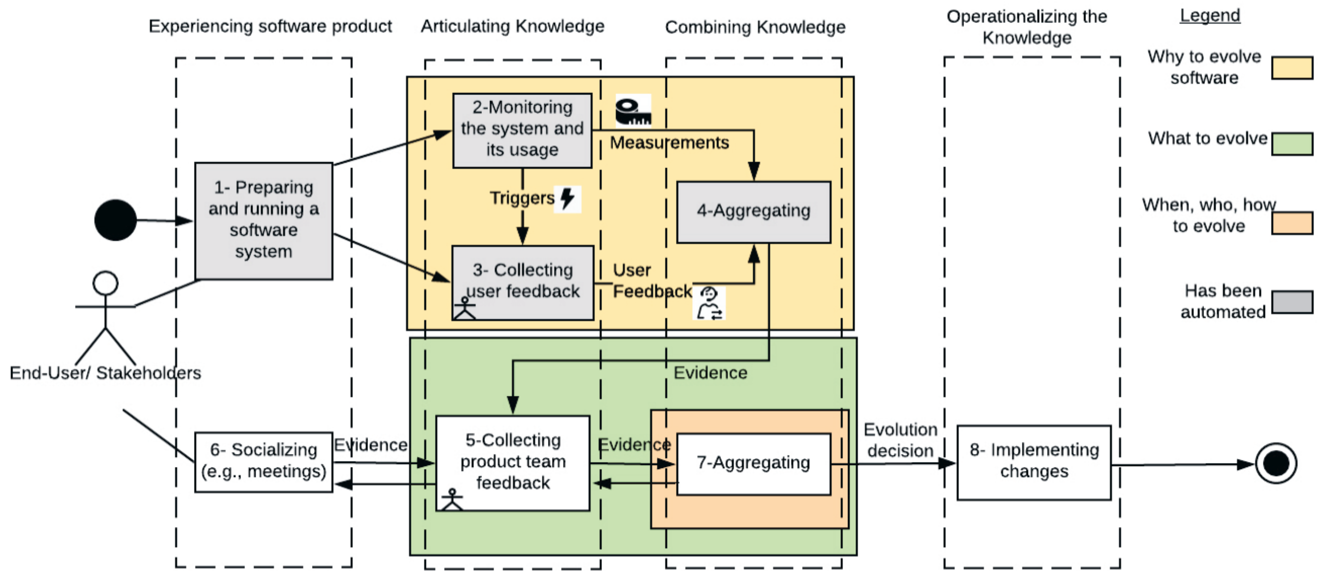


Figure 8-2. GESU method—dashed boxes identify knowledge transformation processes and solid boxes show their activities

Proactive user feedback allows the product team to connect user feedback with the context directly. An autonomous feedback request enables them to find explanations for a situation in a more straightforward manner. Also, a proactive feedback request allows a user to share an immediate perception rather than one that is delayed. Furthermore, this type of feedback request may increase the volume of received feedback as some users do not provide feedback unless requested.

4- Aggregating (measurements and user feedback). This activity aggregates acquired knowledge from Activities 2 and 3. Software system measurements are translated into knowledge for the product team. The user feedback also contains more knowledge for the product team (i.e., users' experiences, perceptions and needs). Combining the two knowledge sources generates new knowledge including evidence for software evolution. Previous studies (Fotrousi et al. 2014; Oriol et al. 2018) have provided technical solutions on how to match the corresponding measurements and user feedback. Supporting the proposed solutions, the GESU suggests forming the *evidence* which not only includes the individual pieces of knowledge and the matching information but also the aggregated forms of the knowledge. Creating the aggregated knowledge may need a level of human activity for synthesizing the prior knowledge (Nonaka and Toyama 2003). The aggregation can be in different formats such as a simple list of evidence, descriptive statistics (Fotrousi et al. 2014), a correlation analysis (Oriol et al. 2018) or a visualization map (which is presented in Section 6.2.2).

5- Collecting product team feedback. In this activity, the product team receives the compiled *evidence*, including the user feedback, system measurements, and the combination of them. Each team member synthesizes the received knowledge together with personal observations and knowledge. Conclusions are made regarding the knowledge as a list of evidence for software evolution that is shared for decision-making.

6- Socializing. Sometimes product team members need to share their perspectives and possibly discuss the evidence for evolving the software. This knowledge sharing is performed via socialization activities such as meetings.

7- Aggregating (product team feedback with previous evidence). The decision-maker combines the synthesis of the product team with the existing knowledge. Activities 5, 6 and 7 are iteratively repeated to collect knowledge from the involved product team members, such as developers and their manager.

8- Implementing changes. The planning knowledge can guide the implementation of the new product evolution.

Activities 2, 3, and 4 involve identifying symptoms to explain the problems and clarify the reasons for particular changes to answer the question: why do we need this change? Activities 6 and 7 identify causes for the symptoms to answer what should be changed. Activity 7 is the only one that involves planning to identify who, when and how to evolve the system. The categories of knowledge define the kinds of changes that happen in the software evolution (Taentzer et al. 2019). In this research, we developed Activities 1–4 (grey boxes in Figure 8-2) to automatically collect knowledge from the end-users and the system.

5. Research Methodology

We followed a design science research approach (Hevner et al. 2004) in which we designed the GESU method and evaluated it using a case study. Figure 8-3 positions our design science research in consideration of its relations with the environment and knowledge base. We applied the acquired knowledge from the SECI theory as well as the concepts and technical solutions that FAME proposed to design the GESU and later used the GESU to revisit the knowledge. Also, we received requirements from the case environment and evaluated the GESU by applying it to the case.

Figure 8-4 provides an overview of our research methodology process that slightly adapted the design science process-model proposed by Peffers et al. (2007). In the figure, we replaced "demonstration" with "Apply" to avoid confusion in the terminology. Also, we merged the process of "define objective and solution" with the "Design" process. In the research process, we investigated the problem from previous studies and designed the GESU method with the support of various theories. We applied the method in a smart parking use case and evaluated the method with the case

For applying and evaluating the GESU, we used a single case study method (Yin 2014). With single case study research, we can test the technical applicability and effectiveness of the artefact and also strengthen our theoretical understanding, while deepening our knowledge of the specific case (Ulriksen and Dadalauri 2016). Yin (2014) described that a single case is justified with a well formulated theory in which a clear set of propositions has been specified. A single case can confirm, challenge or extend a theory.

We defined two evaluation research questions:

> *RQ1: How accurately and completely does GESU explain the knowledge creation process in the smart parking case?*
>
> *RQ2: How applicable and useful is the GESU to support the gathering and sharing of knowledge by the product team for the evolution of the smart parking case?*

RQ1 aimed to evaluate whether the knowledge creation process in the smart parking case accurately and completely explained using the underlying concepts defined in the GESU.

RQ2 aimed to evaluate the benefits of having the GESU method in practice. With RQ2, we investigated whether the GESU was an applicable and useful way of gathering evidence from stakeholders such as users and developers to support product evolution decisions. For applicability, we assessed whether the method could be integrated in the technical infrastructure of the case. For usefulness, we evaluated whether the method could constraint the problem it was meant to solve.

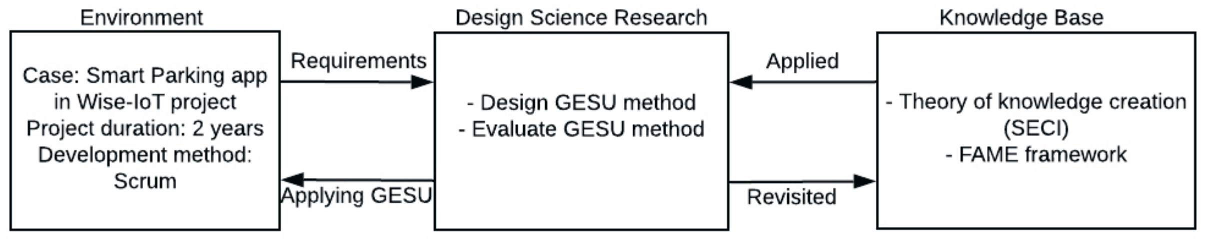


Figure 8-3. Positioning our design science research approach in relation with the environment and knowledge base

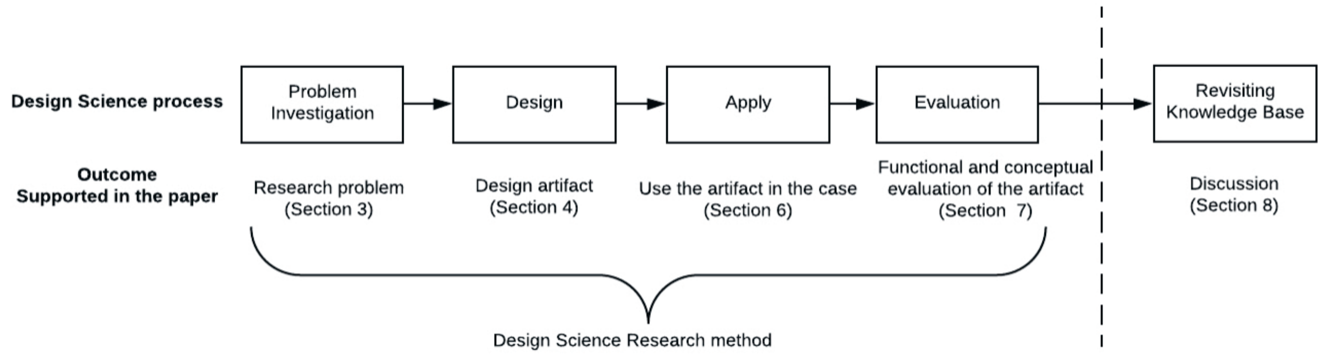

Design Science process
Problem Investigation
Design
Apply
Evaluation
Revisiting Knowledge Base
Outcome Supported in the paper
Research problem (Section 3)
Design artifact (Section 4)
Use the artifact in the case (Section 6)
Functional and conceptual evaluation of the artifact (Section 7)
Discussion (Section 8)
Design Science Research method

Figure 8-4. Research methodology

5.1. Research Context

The study was conducted in the context of a European-Asian innovation project called Wise-IoT (www.wise-iot.eu), where IoT refers to the Internet of Things. The part of the project that we collaborated with in this study was aimed at understanding the problems of developed systems and identifying opportunities to evolve them to increase value creation and the quality of the experience. To achieve this aim, we implemented Activities 1–4 of the GESU to facilitate the gathering of knowledge existing in the use of the developed systems.

For this study, we selected the smart parking software application (Sotres et al. 2018) that the University of Cantabria (UC) had developed for the city of Santander in Spain. The application made use of thousands of IoT traffic and parking sensors that were deployed in the city and helped users find open parking spots within the city when they were travelling by car. The application used a recommender system to generate an unoccupied parking selection and a route to it for end-users.

5.2. Unit of Analysis

The knowledge that was gathered, combined and shared through the GESU methods was the unit of analysis. Part of that knowledge formed the evidence for the software evolution. For example, from user feedback, a developer would learn that a *user is happy* about the application, but *feature x was broken due to a particular action of the user*. The second piece of knowledge may form evidence for changing the software in a future release, while the first one would not. The knowledge could be about the software system reflected in the monitoring data, feedback that users shared about the software product and synthesized knowledge from developers and managers that we captured via interviews.

5.3. Research Process

This section explains our research method process as presented in Figure 8-4.

Problem investigation. We started the research with the problem identification step to understand the challenges regarding existing approaches and to identify the scope of the research. We also studied relevant theories to be used to support the design. In the beginning, we had several meetings and discussions within our team to investigate the problems and possible technical solutions. We explained the identified problems in Section 3.

Design. In this study, we presented one iteration of designing the GESU for the problem. The GESU process and its foundations are explained in Section 4.

Apply. We applied the GESU to a smart parking case piloted in Santander, Spain. The third author instrumented the GESU method (Activities 1–4 in Figure 8-2). The instrument was featured with goal-driven monitoring of user behaviours, autonomous collection of user feedback, and the creation of an insight stream combining the acquired knowledge. For gathering the user feedback, he adapted the user feedback framework from the Supersede Project (www.supersede.eu). The framework provided facilities to customize user feedback forms and configure the triggering mechanisms of the feedback forms. The UC team developing the smart parking application integrated the user feedback framework and the recommender system. They organized the pilot study in Santander. Public meetings for the citizens of Santander interested in the IoT were held. The UC team informed the citizens about the evolution of Santander as a smart city and gave an overview of the most relevant projects, including Wise-IoT. The smart parking application was presented, and the citizens could volunteer for the pilot study, given the prerequisite that they had an Android phone and a car. There was no renumeration for participants who agreed to test the application. A total of 41 citizens had registered and took part in the pilot study. The citizens volunteered because of intrinsic motivation to help the city's evolution as a smart city. The pilot study lasted three months, from the end of February to the end of May, 2018. Within that time, we recorded the insight stream aggregating data from the system monitoring and user feedback. Every month, we had an intermediate manual analysis of the insight stream to check on minor evolutions. When the pilot phase ended, we fully analysed the insight stream.

Evaluation. The evaluation had two parts and each contributed to answer one of the RQs. Interviews in combination with observation were the primary means of data collection. To evaluate the design, we planned one interview each with three developers and one decision-maker of the smart parking application.

The first interviewee was the developer of the smart parking application, who led the development at the UC. She actively participated in collecting user feedback and the monitoring data. The fourth author had an informal interview with her to capture her knowledge based on the user feedback and her knowledge about the sensors. The second interviewee was the developer of the recommender system. The third interviewee was an infrastructure manager, mainly for managing the sensor infrastructure in the city of Santander. He was sometimes involved in the development. The fourth interviewee was a decision-maker in Santander's municipality who was involved in decisions at the Santander site (for example, whether and how a sensor should be changed). Each interview took around 30 minutes on average, and all the interviews were recorded after getting consent from the interviewees. We used semi-structured interviews and engaged the interviewees in a dialogue. The first author was responsible for all except the first interview.

The interviews (except for the first one) started with a brief explanation of the interview's goal and current research. Then the interviewer received permission for recording the interview in order to transcribe it later. To reduce interviewee tension, she asked some warm-up questions regarding the experience and the role of the interviewee in the use case. The second and third interviews started by reviewing the purpose of the software application developed during the use case as well as how user feedback had been collected. The interviewer presented the results of the user feedback analysis to the interviewee, including the synthesized list of user feedback and the frequency of each feedback item. Two questions were formulated to ask whether the interviewees had learnt something from the feedback and whether the learning confirmed for them things they had suspected. Later, the interviewer presented a map showing user feedback associated with a particular sensor location on the map. She repeated the two questions above, asking whether the interviewees had learnt something extra from the map and whether the learning confirmed for them what they had suspected. Later, she continued with questions to understand what the interviewees had done with the knowledge. She also sought to identify how the knowledge was transferred and shared, in what format and whether anybody else was involved in the activity. The interviewer asked further questions for clarification in case the interviewee did not provide a clear answer. In the last interview, with the decision-maker, the interviewer recalled the purpose of the software application developed during the use case. The main focus of the interview was on understanding what knowledge was needed for making a decision, who was involved in the decision-making, how a decision was made, and in which format. Follow-up questions were asked when further clarification was needed.

For the analysis of the interviews, we transcribed them and used a deductive content analysis approach (Elo and Kyngäs 2008) to codify the transcriptions. We used explanation building (Yin 2014) to iteratively check the conformance of the interview data with the method. The deductive approach uses initial coding categories, which were extracted from the theoretical concepts applied to the GESU with the possibility of extending the codes (Hsieh and Shannon 2005).

We conducted the deductive analysis using the three steps described below.

Step 1 – Development of the analysis matrix. We developed a matrix to connect the participants' quotes and the initial categories of codes. The connections were filled with the coding data provided in Step 2. We used an unconstrained matrix with the possibility to extend the categories during the data coding. We expected that the interviewee would provide data regarding the categories associated with the GESU's underlying concepts: process (socialization, internalization, combination and externalization); knowledge (tacit and explicit); knowledge chain (old knowledge and new knowledge); knowledge category and spiral.

Step 2 – Data coding. We reviewed all the quotes from the interviews and coded them in relevance to the defined categories from Step 1. Although we aimed for an unconstrained matrix, we did not recognize new categories during the coding.

Step 3 – Conceptual testing. The coded matrix was a good tool for testing the GESU. Exploring the codes identified the extent to which the case could describe the underlying concepts of the GESU.

5.4. Threats to Validity

We identified and classified the threats to validity and the reliability in the case study (Yin 2014) as described in the following.

The *construct validity* reflects whether a study measures what was supposed to be measured. To increase construct validity, we used two sources for collecting the data: observing the system at runtime with the direct participation of one developer in the data collection and interviewing other members of the product team. We performed triangulation of the interviewees' perspectives with two developers, one infrastructure manager and one product manager to evaluate the method. Furthermore, for the analysis of the interviews, we used investigator triangulation in which the first and second authors analysed and reviewed the analysis, respectively. To increase the construct validity, during the content analysis of the interviews, we established a chain of evidence to ensure that the categories were defined correctly. We also reported the analysis by reporting quotes as appropriate.

One limitation could be relevant to the short duration of the pilot period which limited the ability to observe actual steps in the product evolution. A longer testing period would have allowed for the ability to collect more evidence and include major software updates. This study did not intend to evaluate adherence of next evolutions with the collected evidence.

Internal validity is a concern in an explanatory case study as to whether confounding factors bias the explanation and analysis of the relations. To avoid such threats, we used an explanation-building tactic that provided a pattern for considering the correct inferences and analysed the data accordingly.

Another issue relevant to internal validity was the choice of participants in the case, who experienced the system and provided feedback. The case only engaged with participants who were interested in the innovative application. The participants could have been motivated to provide feedback just because they knew that they were part of a study. There is the possibility that the participants were friendlier than the average user. However, looking at the feedback ratings, we saw that they were not hesitant to give the lowest ratings (one star) when they encountered a problem with a recommendation.

External validity concerns the ability to generalize the results obtained from a study to other situations and contexts.

Generalization was not the goal of this study. A large number of software applications with a variety of product goals, contexts, stakeholders, and company policies for managing knowledge exist. The outcome with respect to applicability and usefulness of the method might differ in another context. However, we could discuss the context in which the product's goal was defined based on the user behaviour, how the users at runtime were able to experience the product and the general benefits from the GESU. As such, the method can be applied in other contexts as long as there are users that have a positive attitude for giving feedback.

Reliability is about the rigor and honesty of the research. Threats to reliability affect the repeatability of the study. To address these threats, we established a study protocol, collected all data in a database, transcribed the interviews and used triangulation as the primary strategy for answering the research questions. We developed detailed coding rules that ensured that the other researchers would make the same decisions. The first author extracted and analysed the codes, and the second author reviewed the analysis performed by the first author.

6. Application of the GESU in the Smart Parking Case

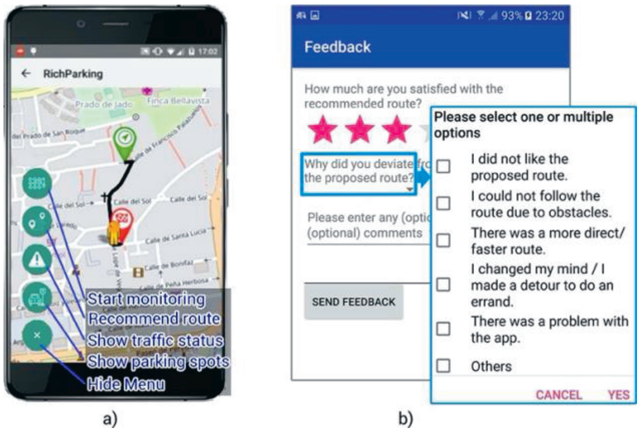
We applied the GESU to the smart parking prototype application for Android smartphones, created by the developers from the University of Cantabria (UC).

6.1. Data Collection

Figure 8-5a shows a screenshot of the Android application. The application used a recommender system to generate an unoccupied parking spot selection and a route to it for the end-users. The parking spot sensors provided data about the spots' current states (open/occupied) and allowed the app to display the open spots. Some of the streets contained sensors to measure traffic load and allowed the app to recommend routes that avoided traffic jams. Each recommendation consisted of an open parking spot and a fast route to the spot.

We instrumented the GESU and the UC team integrated it with the smart parking application. The instrument monitored user behaviour based on data from IoT devices to detect whether users were in a particular situation, such as having achieved a goal or having deviated from the pathway towards the goal. With this approach, requests for user feedback were issued, making the feedback requests relevant for the concerned users and reducing our dependency on luck for useful feedback to be received. The aggregated monitoring data and user feedback were recorded in an insight stream.

The implementation of the GESU parametrized the insight stream to reveal issues with the smart parking application (e.g., usability problems, missing functionality), the parking and pathway recommender system (e.g., bad recommendations, slow performance), the behaviour of the physical IoT devices (e.g., sensors delivering wrong values) and third-party software systems (e.g., outdated street map data).



Feedback

How much are you satisfied with the recommended route?

Why did you deviate from the proposed route?

Please enter any (optional) comments

SEND FEEDBACK

Please select one or multiple options

- ☐ I did not like the proposed route.
- ☐ I could not follow the route due to obstacles.
- ☐ There was a more direct/ faster route.
- ☐ I changed my mind / I made a detour to do an errand.
- ☐ There was a problem with the app.
- ☐ Others

CANCEL YES

Figure 8-5. Screenshots of (a) the smart parking application and (b) a feedback form.

To detect particular situations, our system mapped the recommendation to a goal tree and monitored the user's adherence to the recommendation. The parking spot was the main goal, and each street segment of the route was a sub-goal. When the users' GPS location matched with a street segment, the corresponding sub-goal was set to fulfilled. Previous sub-goals that were not already marked as fulfilled were marked as skipped. We specified the interesting situations as being those when sub-goals were skipped, the user achieved the main goal or abandoned it. In these situations, our goal monitor issued a feedback request tailored to the situation.

Based on the situation, the recommender system selected the feedback to be gathered from the user. One of the following three feedback forms was then displayed:

- *Type 1*. The user deviated from at least 50% of the pathway. The feedback form asked about the user's satisfaction with the recommended route (star rating) and the reasons for the deviation (multiple choice and free text answer). See Figure 8-5b.
- *Type 2.* The user adhered to the pathway but selected a parking spot other than the recommended one. The feedback form asked about the user's satisfaction with the chosen spot

(star rating) and the reasons for not taking the recommended parking spot (multiple choice and free text answer).

- *Type 3.* The users took the recommended spot. The form asked about the user's satisfaction with the recommended route and parking spot (star ratings) and requested a comment with a free text answer.

For the users' safety, we displayed the optional short feedback forms to the user only at the end of a session when the user stopped driving. The user could then rate the experience and provide reasons for the rating. If the user mainly followed the recommended route, the form asked about the user's experience with the parking spot that was taken. Otherwise, the form asked about the user's experience with the route and the reason for the deviation.

For providing anonymity and control of the data collection to the end-user, user sessions were tracked only with a random ID. The user could decide to start the monitoring and whether she or he agreed to send GPS data during the monitoring. The session ended when the user parked the car or cancelled the monitoring. All events, such as the creation of a session, the start and stop of the monitoring, the user feedback and the monitoring result of each location comparison were augmented with a timestamp and written into the insight stream. The structure of the insight stream used the session objects as the top-level entities. This structuring allowed combining the user feedback with the events that occurred in the respective session. For example, when a user provided feedback about a route recommendation, we could compare the feedback with the actual route taken by that user.

The UC managed a pilot study with the citizens of Santander. During the study, monitoring data and user feedback were recorded into a log file and in a database, respectively. The tool created the insight stream combining the acquired knowledge and logged it in one file per day. Once per month, we manually analysed the log files and shared the main findings with the developers of the smart parking application. The Spanish developers evolved the smart parking software and maintained the IoT-based smart city system. The application and its recommender system received two minor updates during the test phase. When the test phase ended, we analysed the insight stream. The developer who was involved in the development helped us to analyse it and discover potential problems, user needs, new requirements and suggestions. The developers evolved the smart parking application including its recommender and maintained the IoT-based smart city system based on the identified evidence.

6.2. Results

This section presents the results. The presentation of results was mapped to the process explained in the GESU method in Figure 8-4. Sections 6.2.1 to 6.2.3 address the processes of gathering evidence for software evolution, and Section 6.2.4 presents the evidence acquired during the processes.

6.2.1. Monitoring the System and its Usage and Collecting User Feedback (Activities 2 and 3)

A total of 303 sessions were created with recommendations for a parking spot and corresponding route to it. In 68 out of the 303 cases, users started the session monitoring after receiving a recommendation. In 26 out of the 68 monitoring sessions, users allowed the mobile app to send their GPS position to the system, enabling the system's adherence monitoring functionality. In ten of these sessions (38.5%), the users adhered to the recommended route.

In 16 out of the 26 adherence-monitoring-enabled sessions, users submitted a feedback form. Seven of these forms were of Type 1 concerning route deviations; five were Type 2 concerning a parking spot deviation and four were Type 3 concerning a fulfilled recommendation.

We could not track users over multiple sessions due to the session-based privacy mechanism. We counted individual sessions instead. The users allowed the mobile app to send their GPS data in 38.2% of the monitoring sessions (26 out of 68). We considered this to be a good amount because some of the 68 monitoring sessions may have been started for testing the app functionality and not to go somewhere or seek a parking spot.

Of the 26 sessions with user GPS data, we received 16 submitted feedback forms. Of the users who were presented a feedback form, 61.5% decided to fill it out and submit it. If we consider all the sessions in which the monitoring was started, the ratio was 16/68 = 23.5%. Again, some of the sessions may have been started for testing the app functionality. We consider the 23.5% feedback ratio to be high in comparison to the use of web surveys (Fan and Yan 2010). The reason for this result could be that we kept the forms small and simple, and they were presented to the users in situations and with content that was relevant to them.

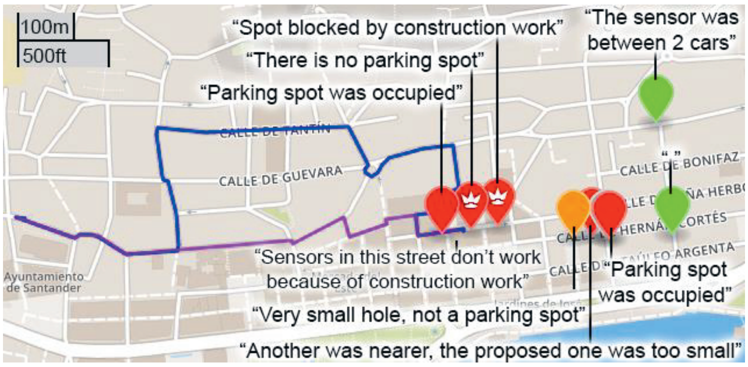
Table 8-1. Categorised user feedback from the free-text answers.

User Feedback	Number of Answers
The parking spot was occupied	6
There was a more direct or faster route	5
The parking spot was too small or the sensor was in a bad location	5
The route or parking spot was blocked by construction work	2
The app was too slow or stalled	2
Found a free parking spot before arriving at the recommended one	1

The average user satisfaction rating of the parking spots was 2.13, and the satisfaction of the recommended routes was 2.07 out of 5 stars. We analysed the automatically collected user feedback for reasons why the scores were not higher and coded the free-text answers. Table 8-1 shows the resulting categories and the number of answers in each category.

6.2.2. Combining User Feedback and Monitoring Data (Activity 4)

To put the above feedback into context, we visualised the monitoring data on a map and added the feedback according to the GPS data from the users. Figure 8-6 shows the parking spots rated by the users, together with their feedback. It also shows one of the recommended routes and the corresponding route taken by the user.



100m
500ft
"Spot blocked by construction work"
"There is no parking spot"
"Parking spot was occupied"
"The sensor was between 2 cars"
"Sensors in this street don't work because of construction work"
"Very small hole, not a parking spot"
"Parking spot was occupied"
"Another was nearer, the proposed one was too small"
CALLE DE TANTÍN
CALLE DE GUEVARA
CALLE DE BONIFAZ
Ayuntamiento de Santander

Figure 8-6. Parking spot feedback*

*: red: negative, red with crown: discussed in the text, green: positive. Blue line: a route recommended to a user. Violet line: route taken by that user

Despite the few user feedback items received by the system, important issues could be identified with the analysis of the correlated monitoring and feedback data. One feedback item mentioned a blocked parking spot due to construction work and another a non-existent spot. When looking at the map, these two spots were close to each other (white crown markers). Also, a recommendation led to another spot nearby on the same street, and the user gave the feedback that the sensors on that street did not function because of construction work. The construction work in that street was an issue that was uncovered by combining the user feedback and monitoring data.

The second group of red markers shows an accumulation of parking spots that were either occupied or too small. However, because of the few data points, we could not say whether this was a condition specific to that location or whether this was a more general problem with the parking spots in the city. For example, one of the positively rated spots also received negative feedback (stating that the sensor was between two cars, which means that it was not possible to park there). However, the green markers were both located in less crowded areas of the city, where the chance of finding an open spot was higher (whether it was the recommended one or another one close by).

Furthermore, one user stated that no parking spot was available at the recommended location. However, the user's GPS data showed that the user never was in that location but went somewhere else instead. This observation is an example of how monitoring data can be used to verify the validity of user feedback. It seems that there was a different issue instead (e.g., the user may not have been able to read the map correctly).

6.2.3. Collecting Product Team Feedback and Aggregating (with Previous Evidence) (Activities 5 and 6)

The interviews with the product team revealed that developers used and combined knowledge from different sources to interpret a situation. The results showed that one source of knowledge was not sufficient to understand the problems and act accordingly. Sometimes, developers anticipated an event and therefore expected to receive the relevant user feedback. At other times, they were surprised by the new knowledge received from the user feedback. Combining the user feedback with sensor locations revealed some new knowledge for the interviewees.

We found that interviewees did not necessarily document their explicit knowledge. Sometimes they performed an action based on their own reasoning without documentation. We also found that decision making could be a complex task where several factors could come into consideration. Managers received feedback from different individuals, then combined and compiled them toward a decision. Decision-making was sometimes as simple as confirming a small change in the code or needed detailed discussion of the changes with the municipality and other managers

6.2.4. Evidence for Software Evolution

The analysis surfaced findings with a significant effect on the maintenance and evolution of the smart city system and the Rich Parking application. Individuals in the product team provided a set of reasoning, but product evolution decisions need a combined form of this reasoning.

Construction work. Increasing construction work blocked streets and parking spots during the test phase. The recommender used an external street routing framework that was not updated with the construction work information in a timely fashion. Therefore, the recommender sometimes proposed routes with blocked street segments. The effect of increasing construction work during the trial phase appeared to be larger than the effects of other factors that could have led to improved route recommendations over time. The 38.5% route adherence seemed to be quite good, given the construction work problem as well as possible cases where users might have decided not to follow the proposed route for other reasons. However, this rationale must be taken with care due to the low number of sessions and the fact that the users were aware of participating in a test, which could have biased them to follow the proposed routes eagerly. Furthermore, construction work could also have had a negative effect on the parking spot ratings. If a user gave a bad rating to a parking spot because it was lying inside a construction zone and thus was not reachable, there was still a chance that in a future session, the system would propose a parking spot close to the badly rated spot and still located within the (same) construction zone.

Based on the observations, the interviewees suggested some improvements in the system. One interviewee suggested *keeping the information regarding the construction updated.* Another interviewee proposed a technical suggestion for *adding an exception to the code for the spots blocked by construction work.*

Parking sensors and fluctuations in parking spot availabilities. The system proposed parking spots to users that were unoccupied at the time when the user requested a recommendation. During high traffic, there was a good chance that another vehicle would park on the proposed spot before the user arrived. As a result, users may have experienced arriving at occupied spots and thus gave bad ratings. *Vehicles were sometimes inaccurately placed with regard to the parking sensors* (in the worst case, in the middle of two parking spots which did not trigger the parking sensors). The knowledge from the interviews implied that *upgrading the hardware or updating the software was needed***.** An improvement to the software could be to let the recommender prioritise regions with large numbers of free parking spots or to introduce a reservation system.

These issues pointed out by the insight stream could not be solved during the pilot phase. However, they provided developers with facts to think of how to improve the system.

The ratio of user ratings to available parking spots. While we received nine user ratings about parking spots, the city of Santander contained hundreds of spots. The coverage of the city's parking spaces was relatively low. A broader use of the smart city-generated IoT data is needed to generate insights for the totality of the city.

7. Evaluation

This section provides a detailed analysis to answer the research questions.

7.1. Accuracy and Completeness of the GESU's Conceptual Framework (Answering RQ1)

The analysis of interviews confirmed that the GESU can completely and accurately describe all the processes the stakeholders gathered during elicitation and the use of the knowledge from the smart parking application. The analysis showed that the elicitation of evidence involved socialization, internalization, combination and externalization processes as the GESU suggested. Also, the analysis showed that the processes in the smart parking application could follow the knowledge conversion operators in the sequence as the GESU suggested. The following paragraphs specify how the support or disagreement was reported during the interviews.

The analysis of interviews indicated that a diversity of individuals could contribute to enhancing knowledge about the product in use through the different GESU processes.

Socialization. The new product impacted individuals while they were experiencing the product and socializing their experience. Evidence of feedback from end-users showed that they had gone through the experience of the product to acquire knowledge about the product. The developers also mentioned their experiences with the product and the development of the product:

"I had to do two things. First, I had to provide a recommendation for a parking place based on a set of parking sensors, and secondly, I had to recommend a route based on the user's current location.

A developer mentioned that he had to report to the municipality as the decision-maker via a meeting. The meeting was done using the *socialization* process to transfer the developer's tacit knowledge to the manager's tacit knowledge in order to inform him of the situation.

Externalization. The tacit knowledge acquired via experiencing had to be synthesised and documented. During the interviews, the developers demonstrated that user feedback and sensor data were two sources of knowledge that they used. Such knowledge was explicit and collected using *externalization*. The interviewees expected some of the user feedback, but some were new explicit knowledge for them:

"I did not highly expect [the feedback], but I can agree now as sensors are not deployed in the whole of Santander. That is why it could happen."

"Yes [I know the user feedback], it has not happened before, but during last week, we realized that there was construction work in the area that parking spots were deployed."

The elicited knowledge might have different value for various roles:

"The events in which the parking spot was occupied did not have much information for us as an IoT manager, but it is interesting for the application developer."

The results of the synthesis could be either tacit or explicit. For example, the developers had a list of suggestions in their minds. One developer communicated the suggestions via a meeting:

"The infrastructure is owned by the municipality, and we need permission to do all the work. When sensors are not working, we need to direct this to the municipality. ... we will have a meeting with the municipality team, and share a report (but it is not a document) to say whether the infrastructure is working properly or not."

The other developer articulated the suggestions via an email:

> *"I was in contact with our local partner, [NAME]. She lives in Santander and knows much better than me. Normally, I contact her whenever I had some things to modify. I first see if she agrees whether it is the case and whether it is what I understood.*

After a developer met with the decision-maker, the decision-maker also had to synthesize his tacit knowledge together with his previous knowledge and decide on an action, while considering several other factors:

> *"Decision-making in the municipality is quite a complex task. There are several factors affects our decision, for example, for changing a sensor. Citizens, the economic part, workers, technical parts of service providers, and others. The decision-maker has to look at all the factors and decide accordingly."*

The second developer emailed the top developer, who was the decision-maker, and wrote about his suggestion for a change. The second decision-maker could integrate her tacit knowledge with the developer's explicit knowledge and synthesize it through the *externalization* process in order to permit the developer to evolve the software.

Combination. User feedback and sensor locations were two explicit pieces of knowledge; each of which served as input for the combination process. We combined these two knowledge pieces in a map using *combination*. The combination revealed new knowledge for developers that they were unaware of earlier:

> *"With this feedback [on the map], we can see which spots are blocked with the construction work."*

> *"The information regarding the placement of the sensor between cars helps us a lot when there are a lot of sensors between cars, as we can analyse the whole street as to how to enhance that behaviour."*

Internalization. The results showed that in order to make the product operationalized, sometimes the planning decisions were taken by another authority and sometimes just confirmation from a higher developer was sufficient. Also, sometimes the planning decisions were well documented and sometimes, particularly for small changes, they were not. The documented planning decisions as a part of the knowledge were operationalized through the *internalization* process:

> *"…The last decision is mainly taken by the mayor, although there is usually a management team and many discussions before the decision…. In Wise-IoT, he contacted [the UC responsible person] to allow changing a sensor… In reality, formal letters are prepared explaining all the details."*

A developer also mentioned that he operationalized small changes based on his tacit knowledge:

> *"When we both [me and the top developer] agreed upon anything, I usually simply implemented that and then later documented it in the deliverables. An example is the privacy-aware implementation of Rich Parking… When we agreed upon a protocol, I immediately implemented that and then later on documented this change in the deliverable."*

Sequence of the knowledge processes. Below we explain how the relations between the processes were supported in the case.

Socialization ⇔ Externalization. An end-user sent feedback based on their tacit knowledge achieved through experiencing the product (socialization ⇒ externalization). Also, the developer who had already synthesized knowledge and acquired relevant tacit knowledge, needed to report to the municipality decision maker via a meeting (externalization ⇒ socialization).

Externalization ⇔ Combination. The analysis showed that the relation between externalization and combination was a bi-directional relationship. A developer synthesized his tacit knowledge by consulting with other developers or aggregating with available explicit knowledge such as user feedback and monitoring data (combination ⇒ externalization). The developers synthesized knowledge to propose a solution for the problem and plan accordingly (externalization ⇒ combination).

Combination⇒Internalization. The relation was established when the decision-maker sent planning decisions or confirmation decisions (i.e., explicit knowledge) to be operationalized and learnt by other individuals using internalization operators. The analysis showed that tacit knowledge could also be operationalized. For example, one of the developers applied the changes based on his tacit knowledge without any documented plan (externalization ⇒ internalization). According to the GESU, the developer had skipped Activity 7 of the process model that is relevant to aggregating explicit knowledge (documented solutions and plans).

The internalization ⇒ socialization relation was established when individuals such as a user or a developer accepted a new product to experience it. The individual was informed about an action (e.g., a new product version) in internalization in the sense that he was informed about the new version, accepted and took it for granted. The individual took its tacit knowledge to experience the product in socialization.

Spiral cycle for knowledge creation. Our analysis confirmed that knowledge gathering was a continuous activity. The continuity regarded the repeating of each process in one cycle to enhance knowledge or repeating the whole cycle while evolving the product. For example, one interviewee highlighted the need for the continuous collection of user feedback:

> *"We may need to have some verification to check whether the same problem occurred with multiple users. With ten users, for example, we see that it is a real problem to consider."*

7.2. Applicability and Usefulness of the GESU in Practice (Answering RQ2)

The analysis confirmed that the GESU method was applicable in principle in the smart parking use case, although some modifications were needed. Also, the GESU was found useful to address the problems discussed in Section 3. Using the GESU, we collected evidence for software evolution relevant to the use case as presented in Figure 8-7. The figure shows the explicit knowledge captured. It presents what knowledge was acquired by whom and belonging to which category of knowledge (i.e., goals, facts, symptoms, cause, evolution). We also identified the evidence that demonstrated the knowledge.

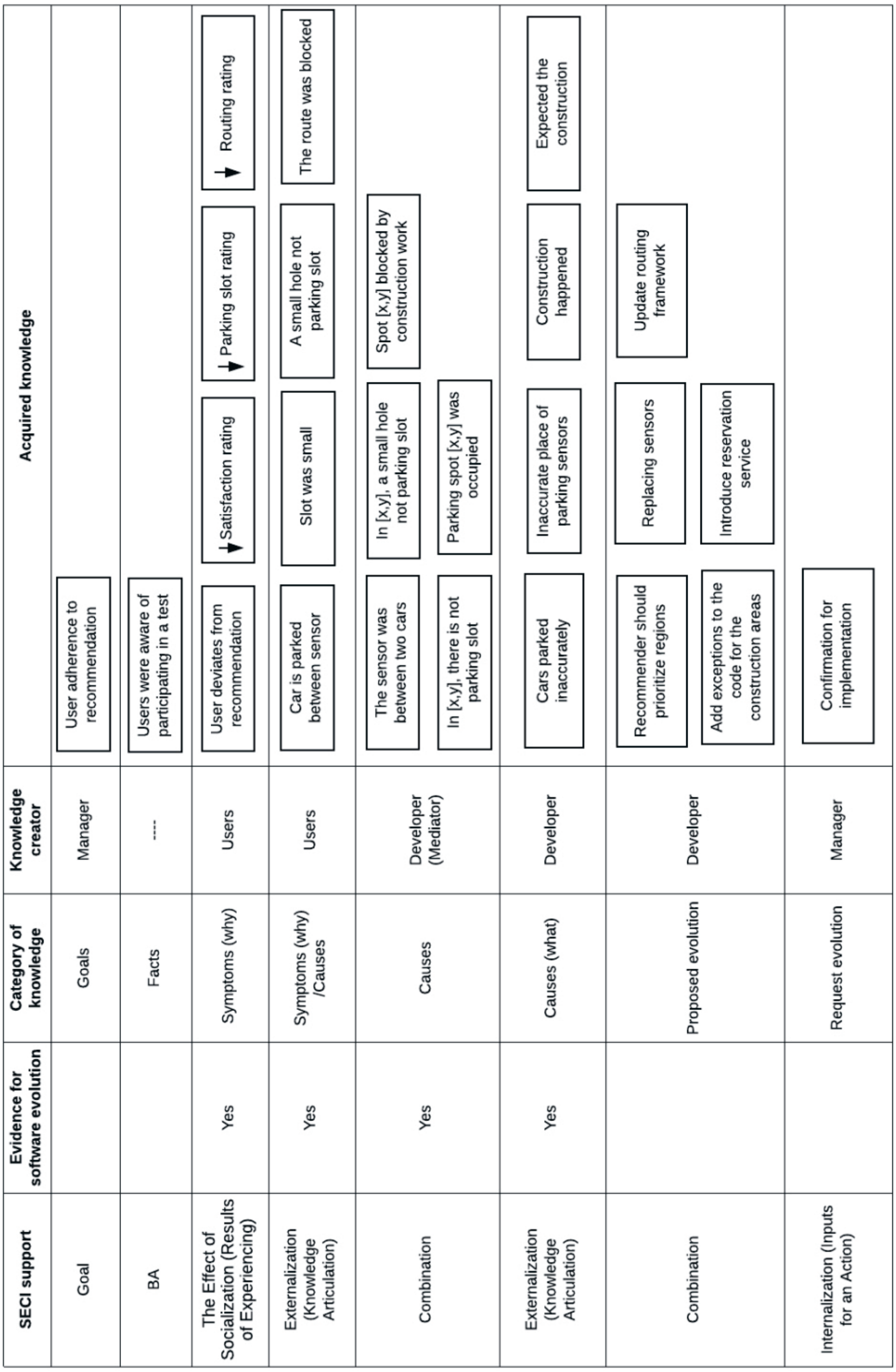
The figure maps the acquired knowledge to the relevant conceptual elements of the SECI. Different knowledge can be grouped together to establish a chain of evidence for supporting the decisions regarding the software evolution.

Applicability: We investigated the applicability of the GESU on the smart parking application following two goals:

- The GESU could succeed to gather evidence from stakeholders to support decision-making in software evolution.
- GESU was technically feasible within the settings of the smart parking application.

Figure 8-7 provides support for the first goal by presenting the knowledge gathered from the stakeholders (users and developers). The knowledge was mainly the explicit knowledge that the stakeholders articulated and could relate to the categories indicating how to support the software evolution. The evidence of why the software evolution is needed demonstrates the symptom knowledge category, and the evidence of what should be evolved shows the cause category of knowledge. We only focused on the evidence that demonstrate the knowledge relevant to the software evolution and ignored other evidence that support knowledge of users or product team acquired in this process, not necessarily relevant to the evolution.

The second goal was also achieved in the study. Our evaluation showed a successful integration of the artefact within the technical infrastructure. The process for monitoring the system, collecting user feedback and combining the data to generate the insight stream was successfully implemented and used in the smart parking use case. The settings for filtering the monitoring data based on goals and triggering the user feedback on the deviation of goals could be also implemented and applied in the case.



SECI support	Evidence for software evolution	Category of knowledge	Knowledge creator	Acquired knowledge			
Goal		Goals	Manager	User adherence to recommendation			
BA		Facts	----	Users were aware of participating in a test			
The Effect of Socialization (Results of Experiencing)	Yes	Symptoms (why)	Users	User deviates from recommendation	↓ Satisfaction rating	↓ Parking slot rating	↓ Routing rating
Externalization (Knowledge Articulation)	Yes	Symptoms (why) /Causes	Users	Car is parked between sensor	Slot was small	A small hole not parking slot	The route was blocked
Combination	Yes	Causes	Developer (Mediator)	The sensor was between two cars In [x,y], there is not parking slot	In [x,y], a small hole not parking slot Parking spot [x,y] was occupied	Spot [x,y] blocked by construction work	
Externalization (Knowledge Articulation)	Yes	Causes (what)	Developer	Cars parked inaccurately	Inaccurate place of parking sensors	Construction happened	Expected the construction
Combination		Proposed evolution	Developer	Recommender should prioritize regions Add exceptions to the code for the construction areas	Replacing sensors Introduce reservation service	Update routing framework	
Internalization (Inputs for an Action)		Request evolution	Manager	Confirmation for implementation			

Figure 8-7. Knowledge landscape. Identifying evidence for software evolution

Usefulness. A design artefact is useful when it constraints the problem it was meant to solve. Below we have listed the problem-solution pairs. The analysis showed that the proposed solutions could solve the problems in practice:

Problem. User feedback lacked enough information about the context in which the feedback was given.

Solution. The method suggested proactive, autonomous requests for user feedback when an interesting situation in the use of a system was detected.

The results clearly stated the context of each feedback. For example, it was clear for each feedback what recommended parking slot or route the user was following. Therefore, the sensors that the user faced on the way were also detectable, which guided finding problematic ones.

Problem. Monitoring data turned out to be difficult to analyse and interpret.

Solution. The method suggested event-driven monitoring based on detecting goal fulfilment.

The results show that it was unnecessary to monitor every single user click in the application and collect volumes of data. The solution suggested monitoring goal deviation (i.e., user deviation from recommendations in the use case) and triggering autonomous feedback to collect more data to explain the user behaviour. The chain of fact extracted from the results and can support the statement. For example, deviation of a user from the recommended parking triggered a feedback form. A user provided the "2/poor" rating for the recommended parking slot and motivated the rating with the statement that "another car parked between two slots".

Problem. No clear understanding of the conceptual process was behind the gathering and organizing of the product knowledge.

Solution. The method was successful in the conceptualization of the gathering and organizing of evidence for software evolution.

The degree of success of this goal was discussed in Section 7-1.

8. Discussion

8.1. Implications

The GESU method theoretically and practically targets gathering, sharing and aggregating evidence to support decisions for software evolution. The GESU provides several benefits. The method steers the collection of user feedback on interesting situations of system use by basing the feedback requests on monitoring the fulfilment of user goals. This technique avoids collecting volumes of unused data and reduces the number of disturbances of users due to feedback requests, by collecting short feedback only when it is needed. Unnecessary disturbances can also negatively impact the amount of feedback received even when it has a negligible impact on the quality of the experience (Fotrousi et al. 2018). Furthermore, in the presented use case, such disturbances could lead to accident. We had chosen to avoid dangerous disturbances and asked users for feedback when they were in a safe situation after the experience.

The evidence that was revealed by combing monitoring data and autonomously gathered user feedback could inform system evolution with advice for new features and how to enhance existing features. For example, better parking sensors that can sense cars or obstacles that are not placed exactly on top of the sensors would be an improvement. This is a solution that takes into account the amount of fluctuation in the available parking spots. Another possibility is a better synchronization of the routing framework with the real-world situation, possibly by connecting the system to the city's database with information about construction work or otherwise blocked streets. The evidence showed what should be changed (the object of the change) and why (the reason for the change). However, the decision-makers need to plan when the change should happen, who should change it and how the software should be changed (Taentzer et al. 2019).

Despite the benefits of the combined analysis of user feedback and monitoring data for software evolution, the analysis also allowed for the identification of two invalid user feedback items for two users who gave feedback about parking spots they never visited. Without the monitoring data, it would not have been possible to distinguish this type of invalid feedback. This result shows that our proposed approach may be used to reduce the risk of collecting irrelevant or even fraudulent feedback (Dalpiaz 2011).

The adaptation of the GESU to the real case shows that the knowledge creation and sharing processes are more interactive than the proposed SECI model in Nonaka's theory (Nonaka and Toyama 2003). There is an interesting implication that machines have a more linear process, while human reasoning needs more back and force interactions. Therefore, it is much easier to automate the process of gathering and combining evidence from user feedback and monitoring data (the upper part of Figure 8-2) as we did in this study, rather than the human reasoning and aggregation of the created knowledge (the lower part of Figure 8-2).

More lightweight approaches, such as questionnaires, may be used as an alternative or complement to elicit requirements (Zowghi and Coulin 2005). With our approach, the real system may be observed in use in the real environment and by real users. This gives the advantage that elicited input such as a feedback may be connected to a specific context, such as a physical location, time or situation in which a system needs modification. Alternative elicitation approaches, such as surveys, would be too short in time, disturb many users and would not allow understanding the context to which the users' feedback pertains.

8.2. Revisiting the Knowledge Base

The GESU method used the SECI model (Nonaka and Toyama 2003) as the theory basis and adjusted the concepts of knowledge sharing from this theory. The findings confirmed that the GESU successfully used the process that the SECI model introduced; however, there were some conceptual disagreements. Therefore, a tailored version of the SECI model was needed to describe the gathering of evidence more accurately. Table 8-2 presents the agreement evaluation of the SECI and GESU.

Table 8-2. Agreement evaluation of the SECI and GESU presented in Figure 8-7.

SECI Features/GESU	**Agreement Evaluation**	**GESU**
Socialization	Agreement	Supported.
Externalization	Agreement	Supported differently. Both tacit and explicit knowledge can be the outcome of the operator.
Combination	Agreement	Supported.
Internalization operator	Agreement	Supported differently. Both tacit and explicit knowledge can be the input of the operator.
Sequence of operators	Disagreement	Internalization was not necessary after combination, and socialization was not necessary before externalization.
Knowledge conversion	Disagreement	Tacit and explicit knowledge between processes might differ.
Spiral cycle	Disagreement	For a product version, a spiral exists within the operators but not for the whole process as the SECI suggested.
Knowledge creation concept	Disagreement	The theory considers internalization as the initial step of knowledge creation, but product impact knowledge was initially created with socialization.

Our analysis confirmed that there was a conversion of tacit to explicit knowledge and vice versa, but in reality, we could not structure our process as Nonaka did (Nonaka and Toyama 2003). As an example, a developer combined the explicit knowledge of feedback with his own tacit knowledge but did not document or externalize the knowledge. Instead, he directly discussed his new undocumented knowledge with his manager.

From the case support, we think that integrating tacit and explicit knowledge can be considered as a part of synthesis using the *externalization* operator. However, integrating two explicit knowledge sources (e.g., user feedback and location data) was performed by the *combination operator*.

The individuals learned about a new version of the product through internalization. We started discussing the model from *socialization*.

The analysis showed that gathering and organizing evidence could not follow the sequence of Nonaka's model and the spiral cycle.

Figure 8-8 presents an update of the SECI model for gathering knowledge in the smart parking application of Santander to show the process sequence. The knowledge gathering and sharing flow in the GESU is relevant to the case and might differ for another case.

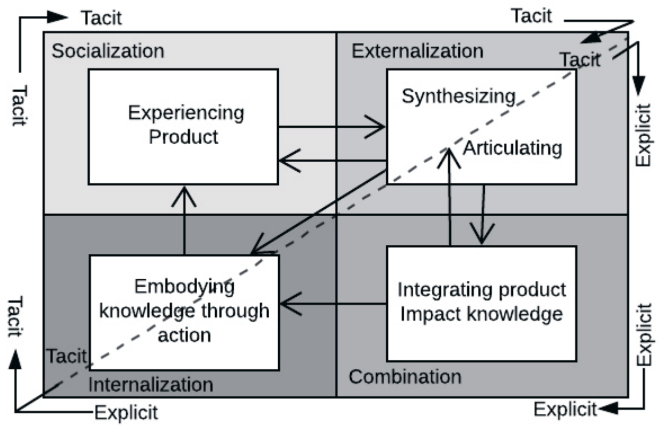


Figure 8-8. SECI model according to the smart city application of Santander.

8.3. Future Work

In the future, GESU should be applied to other cases in different contexts and architectures. The method is particularly helpful for distributed systems and embedded software systems. In such systems, the identification of issues and reasoning for them are more challenging due to the distribution of knowledge among several components in different physical locations (distributed systems) or in various physical devices like sensors (embedded systems). Therefore, the combination of system knowledge and knowledge of the corresponding stakeholders are important to evaluate the achievement of the system's goals.

Gathering evidence for software evolution is more effective if the product team gets in a loop with users: the users provide feedback on their experience of using the product at runtime, and the team requests more feedback on a particular question if needed. Alternatively, the product may self-adapt to improve its functionality in an unsupervised way. A self-adaptive user-feedback acquisition mechanism can trace the effectiveness of collected user feedback and create new feedback requests based on goals (Salehie and Tahvildari 2012), corresponding analytics and perhaps personas (Almaliki et al. 2015) in a timely manner.

Furthermore, standardizing parts of the method allows disseminating the underlying ideas to be used with leading edge techniques. One way could be to implement the insight stream based on IEEE standards for an eXtensible Event Stream (XES_WorkingGroup 2016), which allows for achieving interoperability in event logs and event streams in different systems.

9. Conclusion

This study proposed the GESU method to support decisions for software evolution. The GESU was designed based on knowledge creation theory (SECI). The method combined goal-based system monitoring with proactive, autonomous user-feedback collection. The product system was monitored continuously behind the scenes. The occurrence of an internal event such as deviation of a measurement from its accepted threshold value triggered a request for user feedback. The accepted threshold was defined based on product goals. In response to the triggered request, users shared their perceptions, experience or needs via user feedback that could explain the measurement deviation. The users themselves could also trigger the feedback form and provide similar feedback.

The system measurement was translated into knowledge for the product team. User feedback also contained some more knowledge for the product team (i.e., user's experiences, perceptions and needs). The results showed that combining the two sets of knowledge, aggregated with the product team knowledge, could create knowledge that supported decision-making for system maintenance and evolution.

We implemented the GESU method for a smart city prototype application and interviewed four members of its product team. The initial evaluation showed that the approach was valuable for system evolution: the results were helpful for the smart city Santander partners to adapt and improve their application as well as the IoT sensors deployed in the city. The method provides a systematic approach for gathering user needs, potential issues and new requirements. Such an approach can be especially helpful for distributed systems and the IoT where it is difficult to localize the reasons for potential issues and weaknesses of the system.

Acknowledgement

Part of this work has been supported by the European Union's Horizon 2020 Research and Innovation Programme within the project WISE-IoT under the EU grant agreement No. 723156 and the Swiss SERI grant agreement No. 16.0062.

References

Abelow, D. 1993. Automating Feedback on Software Product Use, CASE Trends December, pp. 15-17.

Adamczyk, P. D., and Bailey, B. P. 2004. "If Not Now, When?: The Effects of Interruption at Different Moments within Task Execution," SIGCHI conference on Human factors in computing systems, Vienna, Austria: ACM.

Ahtinen, A., Mattila, E., Vaatanen, A., Hynninen, L., Salminen, J., Koskinen, E., and Laine, K. 2009. "User Experiences of Mobile Wellness Applications in Health Promotion: User Study of Wellness Diary, Mobile Coach and Selfrelax," 3rd International Conference on Pervasive Computing Technologies for Healthcare, London, UK: IEEE, pp. 1-8.

Ali, R., Dalpiaz, F., Giorgini, P., and Silva Souza, V. 2011. "Requirements Evolution: From Assumptions to Reality," in: BMMDS/EMMSAD. London, UK.

Almaliki, M., Ncube, C., and Ali, R. 2014. "The Design of Adaptive Acquisition of Users Feedback: An Empirical Study," 8th International Conference on Research Challenges in Information Science (RCIS), Marrakech, Morocco: IEEE, pp. 1-12.

Almaliki, M., Ncube, C., and Ali, R. 2015. "Adaptive Software-Based Feedback Acquisition: A Persona-Based Design," Research Challenges in Information Science (RCIS), 2015 IEEE 9th International Conference on: IEEE, pp. 100-111.

Ames, M., and Naaman, M. 2007. "Why We Tag: Motivations for Annotation in Mobile and Online Media," SIGCHI conference on Human factors in computing systems, San Jose, California, USA: ACM, pp. 971-980.

Anandarajan, M., Zaman, M., Dai, Q., and Arinze, B. 2010. "Generation Y Adoption of Instant Messaging: An Examination of the Impact of Social Usefulness and Media Richness on Use Richness," IEEE Transactions on Professional Communication (53:2), pp. 132-143.

Antón, A. I., and Potts, C. 1998. "The Use of Goals to Surface Requirements for Evolving Systems," International Conference on Software Engineering, Kyoto, Japan: IEEE, pp. 157-166.

Antons, J.-N., Arndt, S., Schleicher, R., and Möller, S. 2014. "Brain Activity Correlates of Quality of Experience," in Quality of Experience. Springer, pp. 109-119.

Bailey, B. P., Konstan, J. A., and Carlis, J. V. 2001. "The Effects of Interruptions on Task Performance, Annoyance, and Anxiety in the User Interface," IFIP International Conference on Human Computer Interaction (INTERACT), Tokyo, Japan, pp. 593-601.

Barbosa, O., and Alves, C. 2011. "A Systematic Mapping Study on Software Ecosystems," in: The 2nd International Conference on Software Business (ICSOB 2011). Brussels, Belgium: pp. 15-26.

Barrett, L. F., Mesquita, B., and Gendron, M. 2011. "Context in Emotion Perception," Current Directions in Psychological Science (20:5), pp. 286-290.

Bass, L., Clements, P., and Kazman, R. 2012. Software Architecture in Practice, (3rd ed.). Addison-Wesley Professional.

Bekkers, W., van de Weerd, I., Spruit, M., and Brinkkemper, S. 2010. "A Framework for Process Improvement in Software Product Management," in Systems, Software and Services Process Improvement. Springer, pp. 1-12.

Beyer, J., and Möller, S. 2014. "Gaming," in Quality of Experience. Springer, pp. 367-381.

Bider, I., and Jalali, A. 2016. "Agile Business Process Development: Why, How and When—Applying Nonaka's Theory of Knowledge Transformation to Business Process Development," Information Systems and e-Business Management (14:4), pp. 693-731.

Bjørnson, F. O., and Dingsøyr, T. 2008. "Knowledge Management in Software Engineering: A Systematic Review of Studied Concepts, Findings and Research Methods Used," Information and Software Technology (50:11), pp. 1055-1068.

Blank, S. 2013. "Why the Lean Start-up Changes Everything," Harvard Business Review (91:5), pp. 63-72.

Boegh, J. 2008. "A New Standard for Quality Requirements," IEEE Software (25:2), pp. 57-63.

Boley, H., and Chang, E. 2007. Digital Ecosystems: Principles and Semantics.

Bosch, J. 2012. "Building Products as Innovation Experiment Systems," in: International Conference on Software Business (ICSOB 2012). Cambridge, MA, USA.

Braz, C., Seffa, A., and M'Raihi, D. 2007. "Designing a Trade-Off between Usability and Security: A Metrics-Based Model," in: 11th IFIP TC 13 International Conference on Human-Computer Interaction (INTERACT 2007). Rio de Janeiro, Brazil.

Brill, O., and Knauss, E. 2011a. "Structured and Unobtrusive Observation of Anonymous Users and Their Context for Requirements Elicitation," 19th International Conference on Requirements Engineering (RE) Trento, Italy: IEEE.

Brill, O., and Knauss, E. 2011b. "Structured and Unobtrusive Observation of Anonymous Users and Their Context for Requirements Elicitation," 19th International Conference on Requirements Engineering (RE) Trento, Italy: IEEE, pp. 175-184.

Brinkkemper, S. 1996. "Method Engineering: Engineering of Information Systems Development Methods and Tools," Information and software technology (38:4), pp. 275-280.

Broekens, J., Pommeranz, A., Wiggers, P., and Jonker, C. M. 2010. "Factors Influencing User Motivation for Giving Online

Preference Feedback," 5th Multidisciplinary Workshop on Advances in Preference Handling (MPREF'10), Lisbon, Portugal: Citeseer.

Brooks, P., and Hestnes, B. 2010. "User Measures of Quality of Experience: Why Being Objective and Quantitative Is Important," Network, IEEE (24:2), pp. 8-13.

Buse, R., and Zimmermann, T. 2010. "Analytics for Software Development," in: Foundations of Software Engineering (FSE)/SDP workshop on Future of software engineering research. Santa Fe, NM, USA: ACM.

Buse, R., and Zimmermann, T. 2012. "Information Needs for Software Development Analytics," in: International Conference on Software Engineering, ICSE 2012. Zurich, Switzerland: IEEE Press, pp. 987-996.

Canale, S., Facchinei, F., Gambuti, R., Palagi, L., and Suraci, V. 2014. "User Profile Based Quality of Experience," 18th Internation Conference on Computers (part of CSCC '14), Santorini Island, Greece.

Carlson, J. R., and Zmud, R. W. 1999. "Channel Expansion Theory and the Experiential Nature of Media Richness Perceptions," Academy of management journal (42:2), pp. 153-170.

Carreño, L., and Winbladh, K. 2013. "Analysis of User Comments: An Approach for Software Requirements Evolution," in: 35th International Conference on Software Engineering (ICSE 2013). San Francisco, CA, USA.

Carrizo, D., Dieste, O., and Juristo, N. 2014. "Systematizing Requirements Elicitation Technique Selection," Information and Software Technology (56:6), pp. 644-669.

Carver, J., Jaccheri, L., Morasca, S., and Shull, F. 2003. "Issues in Using Students in Empirical Studies in Software Engineering Education," in: Ninth International Software Metrics Symposium. IEEE, pp. 239-249.

Chapin, N., Hale, J., Khan, K., Ramil, J., and Tan, W.-G. 2001. "Types of Software Evolution and Software Maintenance," Journal of Software Maintenance and Evolution: Research and Practice (13:1), pp. 3-30.

Chapin, S. F., Torn, M. S., and Tateno, M. 1996. "Principles of Ecosystem Sustainability," American Naturalist), pp. 1016-1037.

Choudhary, V. C. 2007. "Software as a Service: Implications for Investment in Software Development," in: 40th Annual Hawaii International Conference on System Sciences, HICSS'07. Waikoloa, Big Island, Hawaii

Chung, L., Nixon, B., Yu, E., and Mylopoulos, J. 2000. Non-Functional Requirements in Software Engineering. Boston, USA: Springer US.

Clements, P., and Bass, L. 2010. "Using Business Goals to Inform a Software Architecture," in: 18th IEEE International on Requirements Engineering Conference (RE'10). Sydney, NSW, Australia.

Clifton, B. 2012. Advanced Web Metrics with Google Analytics. Wiley. com.

Cokins, G. 2009. Performance Management: Integrating Strategy Execution, Methodologies, Risk, and Analytics. John Wiley & Sons.

Collins, H. 2010. Tacit and Explicit Knowledge. University of Chicago Press.

Cooper, A. 2012. "What Is Analytics? Definition and Essential Characteristics," CETIS Analytics Series (1:5), pp. 1-10.

Cooper, R. G., Edgett, S. J., and Kleinschmidt, E. J. 1999. "New Product Portfolio Management: Practices and Performance," Journal of product innovation management (16:4), pp. 333-351.

Costanza, R. 1992. "Toward an Operational Definition of Ecosystem Health," Ecosystem health: New goals for environmental management), pp. 239-256.

Costanza, R., and Mageau, M. 1999. "What Is a Healthy Ecosystem?," Aquatic ecology (33:1), pp. 105-115.

Côté, N., and Berger, J. 2014. "Speech Communication," in Quality of Experience. Springer, pp. 165-177.

Cusumano, M. A. 2008. "The Changing Software Business: Moving from Products to Services," Computer (41:1), pp. 20-27.

Cysneiros, L. M., and Sampaio do Prado Leite, J. C. 2004. "Nonfunctional Requirements: From Elicitation to Conceptual Models," IEEE Transactions on Software Engineering (30:5), pp. 328-350.

Dąbrowski, J., Kifetew, F. M., Muñante, D., Letier, E., Siena, A., and Susi, A. 2017. "Discovering Requirements through Goal-Driven Process Mining," 25th International Requirements Engineering Conference Workshops (REW): IEEE, pp. 199-203.

Daft, R. L., and Lengel, R. H. 1986. "Organizational Information Requirements, Media Richness and Structural Design," Management science (32:5), pp. 554-571.

Dalpiaz, F. 2011. "Social Threats and the New Challenges for Requirements Engineering," in: 1st International WOrkshop on Requirements Engineering for Social Computing (RESC 2011). Trento, Italy.

Davenport, T. H., and Harris, J. G. 2007. Competing on Analytics: The New Science of Winning. Harvard Business Press.

Davis, F. D., Bagozzi, R. P., and Warshaw, P. R. 1989. "User Acceptance of Computer Technology: A Comparison of Two Theoretical Models," Management science (35:8), pp. 982-1003.

Denne, M., and Cleland-Huang, J. 2004. "The Incremental Funding Method: Data-Driven Software Development," Software, IEEE (21:3), pp. 39-47.

Dennis, A. R., Fuller, R. M., and Valacich, J. S. 2008. "Media, Tasks, and Communication Processes: A Theory of Media Synchronicity," MIS quarterly (32:3), pp. 575-600.

Dennis, A. R., and Kinney, S. T. 1998. "Testing Media Richness Theory in the New Media: The Effects of Cues, Feedback, and Task Equivocality," Information systems research (9:3), pp. 256-274.

Devanbu, P., Zimmermann, T., and Bird, C. 2016. "Belief & Evidence in Empirical Software Engineering," 38th International Conference on Software Engineering (ICSE): IEEE, pp. 108-119.

Dilman, M., and Raz, D. 2002. "Efficient Reactive Monitoring," IEEE journal on selected areas in communications (20:4), pp. 668-676.

Doerr, J., Kerkow, D., Koenig, T., Olsson, T., and Suzuki, T. 2005. "Non-Functional Requirements in Industry-Three Case Studies Adopting an Experience-Based Nfr Method," 13th IEEE International Conference on Requirements Engineering, Paris, France: IEEE, pp. 373-382.

Dyba, T., Kitchenham, B. A., and Jorgensen, M. 2005. "Evidence-Based Software Engineering for Practitioners," IEEE software (22:1), pp. 58-65.

Dzvonyar, D., Krusche, S., Alkadhi, R., and Bruegge, B. 2016. "Context-Aware User Feedback in Continuous Software Evolution," International Workshop on Continuous Software Evolution and Delivery (CSED), Austin, USA: IEEE, pp. 12-18.

Ebert, C. 2007. "The Impacts of Software Product Management," Journal of Systems and Software (80:6), pp. 850-861.

Ebert, C., and Brinkkemper, S. 2014. "Software Product Management–an Industry Evaluation," Journal of Systems and Software (95), pp. 10-18.

Edison, H., Smørsgård, N. M., Wang, X., and Abrahamsson, P. 2018. "Lean Internal Startups for Software Product Innovation in

Large Companies: Enablers and Inhibitors," Journal of Systems and Software (135), pp. 69-87.

Eisenmann, T. R., Ries, E., and Dillard, S. 2012. "Hypothesis-Driven Entrepreneurship: The Lean Startup," Harvard Business School Entrepreneurial Management Case:812-095).

Elling, S., Lentz, L., and Jong, M. d. 2012. "Users' Abilities to Review Web Site Pages," Journal of business and technical communication (26:2), pp. 171-201.

Elo, S., and Kyngäs, H. 2008. "The Qualitative Content Analysis Process," Journal of advanced nursing (62:1), pp. 107-115.

Esterle, L., and Grosu, R. 2016. "Cyber-Physical Systems: Challenge of the 21st Century," e & i Elektrotechnik und Informationstechnik (133:7), pp. 299-303.

Fabijan, A., Olsson, H. H., and Bosch, J. 2016. "Time to Say'good Bye': Feature Lifecycle," 2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA): IEEE, pp. 9-16.

Fan, W., and Yan, Z. 2010. "Factors Affecting Response Rates of the Web Survey: A Systematic Review," Computers in Human Behavior (26:2), pp. 132-139.

Feiten, B., Garcia, M.-N., Svensson, P., and Raake, A. 2014. "Audio Transmission," in Quality of Experience. Springer, pp. 229-245.

Fernández-Dols, J.-M., and Russell, J. A. 2003. "Emotions, Affects, and Mood in Social Judgements," in Handbook of Psychology, Personality and Social Psychology, T. Millon, M.J. Lerner and I.B. Weiner (eds.). Hoboken New Jersey: John Wiley & Sons, pp. 283-297.

Ferry, D. L., Kydd, C. T., and Sawyer, J. E. 2001. "Measuring Facts of Media Richness," Journal of Computer Information Systems (41:4), pp. 69-78.

Fickas, S., and Feather, M. 1995. "Requirements Monitoring in Dynamic Environments," in: 2nd International Symposium on Requirements Engineering (RE'95). York, U.K.

Fiedler, M., and Hoßfeld, T. 2010. "Quality of Experience-Related Differential Equations and Provisioning-Delivery Hysteresis," 21st ITC Specialist Seminar on Multimedia Applications-Traffic, Performance and QoE Miyazaki, Japan.

Fiedler, M., Hossfeld, T., and Tran-Gia, P. 2010. "A Generic Quantitative Relationship between Quality of Experience and Quality of Service," IEEE Network (24:2), pp. 36-41.

Forbrig, P. 2017. "Does Continuous Requirements Engineering Need Continuous Software Engineering?," Requirements Engineering: Foundation for Software Quality (REFSQ), Essen, Germany: Springer.

Fotrousi, F., and Fricker, S. A. 2016. "Qoe Probe: A Requirement-Monitoring Tool," 22nd Internation Conference on Requirement Engineering: Foundation for Software Quality (REFSQ), Gothenburg, Sweden: Springer.

Fotrousi, F., Fricker, S. A., and Fiedler, M. 2014. "Quality Requirements Elicitation Based on Inquiry of Quality-Impact Relationships," 22nd International Conference on Requirements Engineering, Karlskrona, Sweden: IEEE, pp. 303-312.

Fotrousi, F., Fricker, S. A., and Fiedler, M. 2018. "The Effect of Requests for User Feedback on Quality of Experience," Software Quality Journal (26:2), pp. 385-415.

Fotrousi, F., Izadyan, K., and Fricker, S. A. 2013. "Analytics for Product Planning: In-Depth Interview Study with Saas Product Managers," in: IEEE 6th International Conference on Cloud Computing. Santa Clara Marriott, CA, USA.

Fotrousi, F., Seyff, N., and Börstler, J. 2017. "Ethical Considerations in Research on User Feedback," 25th International Requirements Engineering Conference Workshops (REW), Lisbon, Portugal: IEEE, pp. 194-198.

Fowler, F. J. 2009. Survey Research Methods. Sage.

Fricker, S., and Schumacher, S. 2012. "Release Planning with Feature Trees: Industrial Case," in Requirements Engineering: Foundation for Software Quality. Springer, pp. 288-305.

Fricker, S. A., and Glinz, M. 2010. "Comparison of Requirements Hand-Off, Analysis, and Negotiation: Case Study," in: 18th IEEE International Requirements Engineering Conference (RE'10). Sydney, Australia.

Fricker, S. A., Gorschek, T., Byman, C., and Schmidle, A. 2010. "Handshaking with Implementation Proposals: Negotiating Requirements Understanding," IEEE Software (27:2), pp. 72-80.

Fricker, S. A., Maedche, A., Botzenhardt, A., and Neer, L. 2012. "Software Product Management," in Software for People. Springer Berlin Heidelberg, pp. 53-81.

Fricker, S. A., Schneider, K., Fotrousi, F., and Thuemmler, C. 2015. "Workshop Videos for Requirements Communication," Requirements Engineering), pp. 1-32.

Froehlich, J., Chen, M. Y., Consolvo, S., Harrison, B., and Landay, J. A. 2007. "Myexperience: A System for in Situ Tracing and Capturing of User Feedback on Mobile Phones," Proceedings of 5th international conference on Mobile systems, applications and services (MobiSys), San Juan, Puerto Rico.

Gallivan, M. J., and Keil, M. 2003. "The User–Developer Communication Process: A Critical Case Study," Information Systems Journal (13:1), pp. 37-68.

Garcia, M.-N., Argyropoulos, S., Staelens, N., Naccari, M., Rios-Quintero, M., and Raake, A. 2014. "Video Streaming," in Quality of Experience. Springer, pp. 277-297.

Garland, R. 1991. "The Mid-Point on a Rating Scale: Is It Desirable," Marketing bulletin (2:1), pp. 66-70.

Gilb, T. 2005. Competitive Engineering: A Handbood for Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage. Butterworth-Heinemann.

Glinz, M. 2005. "Rethinking the notion of Non-Functional Requirements," in: 3rd World Congress for Software Quality. Munich, Germany.

Glinz, M. 2007. "On Non-Functional Requirements," in: IEEE International Requirements Engineering Conference (RE'07). New Delhi, India.

Godfrey, M. W., and German, D. M. 2008. "The Past, Present, and Future of Software Evolution," 2008 Frontiers of Software Maintenance: IEEE, pp. 129-138.

Golafshani, N. 2003. "Understanding Reliability and Validity in Qualitative Research," 1052-0147, pp. 597-606.

Golaszewski, S. 2013. "Flexisketch." from https://play.google.com/store/apps/details?id=ch.uzh.ifi.rerg.flexisketch&hl=en

Gold, N., Mohan, A., Knight, C., and Munro, M. 2004. "Understanding Service-Oriented Software," Software, IEEE (21:2), pp. 71-77.

Goldsby, H. J., Sawyer, P., Bencomo, N., Cheng, B. H., and Hughes, D. 2008. "Goal-Based Modeling of Dynamically Adaptive System Requirements," 15th International Conference and Workshop on the Engineering of Computer-based Systems (ECBS), Belfast, Northern Ireland: IEEE, pp. 36-45.

Gorchels, L. 2000. The Product Manager's Handbook: The Complete Product Management Resource. NTC Business Books Illinois, USA.

Gottesdiener, E. 2002. Requirements by Collaboration: Workshops for Defining Needs. Addison-Wesley Professional.

Guest, G., Bunce, A., and Johnson, L. 2006. "How Many Interviews Are Enough? An Experiment with Data Saturation and Variability," Field methods (18:1), pp. 59-82.

Guzman, E., Bhuvanagiri, P., and Bruegge, B. 2014. "Fave: Visualizing User Feedback for Software Evolution," Software Visualization (VISSOFT), 2014 Second IEEE Working Conference on: IEEE, pp. 167-171.

Guzman, E., El-Haliby, M., and Bruegge, B. 2015. "Ensemble Methods for App Review Classification: An Approach for Software Evolution (N)," 30th IEEE/ACM International Conference on Automated Software Engineering (ASE): IEEE, pp. 771-776.

Guzman, E., Ibrahim, M., and Glinz, M. 2017. "A Little Bird Told Me: Mining Tweets for Requirements and Software Evolution," 25th International Requirements Engineering Conference (RE): IEEE, pp. 11-20.

Guzman, E., and Maalej, W. 2014. "How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews," 22nd International Requirements Engineering Conference (RE), Karlskrona, Sweden: IEEE, pp. 153-162.

Guzmán, L., Oriol, M., Rodríguez, P., Franch, X., Jedlitschka, A., and Oivo, M. 2017. "How Can Quality Awareness Support Rapid Software Development?–a Research Preview," International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ), Essen, Germany: Springer, pp. 167-173.

Hacker, S., and Von Ahn, L. 2009. "Matchin: Eliciting User Preferences with an Online Game," Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: ACM, pp. 1207-1216.

Haigh, M. 2010. "Software Quality, Non-Functional Software Requirements and It-Business Alignment," Software Quality Journal (18:3), pp. 361-385.

Hair, J. F., Risher, J. J., Sarstedt, M., and Ringle, C. M. 2019. "When to Use and How to Report the Results of Pls-Sem, European Business Review.

Hair Jr, J. F., Hult, G. T. M., Ringle, C., and Sarstedt, M. 2016. A Primer on Partial Least Squares Structural Equation Modeling (Pls-Sem). Sage publications.

Hak, T., and Dul, J. 2009. "Pattern Matching," in: Encyclopedia of case study research, A.J. Mills, G. Durepos and E. Wiebe (eds.). Thousands of Oaks, CA: Sage Publications, pp. 663-665.

Hansen, B. H., and Kautz, K. 2004. "Knowledge Mapping: A Technique for Identifying Knowledge Flows in Software Organisations," European Conference on Software Process Improvement: Springer, pp. 126-137.

Hartson, H. R., and Castillo, J. C. 1998. "Remote Evaluation for Post-Deployment Usability Improvement," Proceedings of the working conference on Advanced visual interfaces, pp. 22-29.

Hassenzahl, M., Wessler, R., and Hamborg, K.-C. 2001. "Exploring and Understanding Product Qualities That Users Desire," 5th Annual Conference of the Human-Computer Interaction Group of the British Computer Society (IHm-HCI 01), Lille, France, pp. 95-96.

Heller, F., Lichtschlag, L., Wittenhagen, M., Karrer, T., and Borchers, J. 2011. "Me Hates This: Exploring Different Levels of User Feedback for (Usability) Bug Reporting," in Chi'11 Extended Abstracts on Human Factors in Computing Systems. pp. 1357-1362.

Herrera, M., Moraga, M. Å., Caballero, I., and Calero, C. 2010. "Quality in Use Model for Web Portals (Qiuwep)," in Current Trends in Web Engineering. Springer, pp. 91-101.

Herrmann, A., and Paech, B. 2008. "Moqare: Misuse-Oriented Quality Requirements Engineering," Requirements Engineering (13:1), pp. 73-86.

Herzog, A. R., and Bachman, J. G. 1981. "Effects of Questionnaire Length on Response Quality," Public Opinion Quarterly (45:4), pp. 549-559.

Hess, J., Wan, L., Ley, B., and Wulf, V. 2012. "In-Situ Everywhere: A Qualitative Feedback Infrastructure for Cross Platform Home-It," 10th European conference on Interactive tv and video (EuroiTV), Berlin, Germany, pp. 75-78.

Hevner, A. R., March, S. T., Park, J., and Ram, S. 2004. "Design Science in Information Systems Research," MIS quarterly (28:1), pp. 75-105.

Holsapple, C., Lee-Post, A., and Pakath, R. 2014. "A Unified Foundation for Business Analytics," Decision Support Systems (64), pp. 130-141.

Höst, M., Regnell, B., and Wohlin, C. 2000. "Using Students as Subjects—a Comparative Study of Students and Professionals in Lead-Time Impact Assessment," Empirical Software Engineering (5:3), pp. 201-214.

Howard, J. A., and Sheth, J. N. 1969. The Theory of Buyer Behavior. Wiley New York.

Hsieh, H.-F., and Shannon, S. E. 2005. "Three Approaches to Qualitative Content Analysis," Qualitative health research (15:9), pp. 1277-1288.

Iansiti, M., and Richards, G. L. 2006. "Information Technology Ecosystem: Structure, Health, and Performance," Antitrust Bull. (51), p. 77.

IBosch, J. 2009. "From Software Product Lines to Software Ecosystems," in: 13th International Software Product Line Conference (SPLC 2009). San Francisco, CA, USA: Carnegie Mellon University, pp. 111-119.

Ickin, S., Wac, K., Fiedler, M., Janowski, L., Hong, J.-H., and Dey, A. K. 2012. "Factors Influencing Quality of Experience of Commonly Used Mobile Applications," Communications Magazine, IEEE (50:4), pp. 48-56.

Inzinger, C., Hummer, W., Satzger, B., Leitner, P., and Dustdar, S. 2014. "Generic Event-Based Monitoring and Adaptation

Methodology for Heterogeneous Distributed Systems," Software: Practice and Experience (44:7), pp. 805-822.

Irvine, C., and Levin, T. 2000. "Quality of Security Service," in: 2000 Workshop on New Security Paradigms (NSPW'00). New York, NY, USA.

ITU-T. 2003. "Itu-T P.800," in: in Mean Opinion Score(MOS) terminology, ed: Telecommunication Standardization Sector of ITU.

Ivory, M. Y., and Hearst, M. A. 2001. "The State of the Art in Automating Usability Evaluation of User Interfaces," ACM Computing Surveys (CSUR) (33:4), pp. 470-516.

Jacobs, S. 1999. "Introducing Measurable Quality Requirements: A Case Study," in: 4th IEEE International Symposium on Requirements Engineering (RE'99). Limerick, Ireland.

Jansen, S., Finkelstein, A., and Brinkkemper, S. 2009 "A Sense of Community: A Research Agenda for Software Ecosystems," in: 31st International Conference on Software Engineering (ICSE 2009) Vancouver, Canada: IEEE, pp. 187-190.

Jin, X., Wah, B. W., Cheng, X., and Wang, Y. 2015. "Significance and Challenges of Big Data Research," Big Data Research (2:2), pp. 59-64.

Johann, T., and Maalej, W. 2015. "Democratic Mass Participation of Users in Requirements Engineering?," 23rd international requirements engineering conference (RE), Ottawa, Canada: IEEE, pp. 256-261.

Jordan, P. W. 1998. "Human Factors for Pleasure in Product Use," Applied ergonomics (29:1), pp. 25-33.

Jung, G., Hiltunen, M., Joshi, K., Schlichting, R., and Pu, C. 2010. "Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures," in: IEEE International Conference on Distributed Computing Systems (ICDCS 2010). Genoa, Italy.

Karapanos, E. 2013. "User Experience over Time," in Modeling Users' Experiences with Interactive Systems. Springer, pp. 57-83.

Kennedy, M. M. 1979. "Generalizing from Single Case Studies," Evaluation quarterly (3:4), pp. 661-678.

Khan, A., Sun, L., Jammeh, E., and Ifeachor, E. 2010. "Quality of Experience-Driven Adaptation Scheme for Video Applications over Wireless Networks," IET communications (4:11), pp. 1337-1347.

Khirman, S., and Henriksen, P. 2002. "Relationship between Quality-of-Service and Quality-of-Experience for Public Internet Service," 3rd Workshop on Passive and Active Measurement, Fort Collins, Colorado, USA.

Kifetew, F., Munante, D., Perini, A., Susi, A., Siena, A., and Busetta, P. 2017. "Dmgame: A Gamified Collaborative Requirements Prioritisation Tool," 2017 IEEE 25th International Requirements Engineering Conference (RE): IEEE, pp. 468-469.

Kilkki, K. 2008. "Quality of Experience in Communications Ecosystem," Journal of Universal Computer Science (14:5), pp. 615-624.

Kim, H.-J., Lee, D. H., Lee, J. M., Lee, K.-H., Lyu, W., and Choi, S.-G. 2008a. "The Qoe Evaluation Method through the Qos-Qoe Correlation Model," Fourth International Conference on Networked Computing and Advanced Information Management (NCM'08) Gyeongju, Korea: IEEE, pp. 719-725.

Kim, J. H., Gunn, D. V., Schuh, E., Phillips, B., Pagulayan, R. J., and Wixon, D. 2008b. "Tracking Real-Time User Experience (True): A Comprehensive Instrumentation Solution for Complex Systems," SIGCHI conference on Human Factors in Computing Systems, Florence, Italy: ACM.

Kitchenham, B. A., Budgen, D., and Brereton, P. 2015. Evidence-Based Software Engineering and Systematic Reviews. CRC press.

Kitchenham, B. A., Dyba, T., and Jorgensen, M. 2004. "Evidence-Based Software Engineering," Proceedings. 26th International Conference on Software Engineering: IEEE, pp. 273-281.

Kitchenham, B. A., and Pfleeger, S. L. 2008. "Personal Opinion Surveys," in Guide to Advanced Empirical Software Engineering. Springer, pp. 63-92.

Kittlaus, H.-B., and Clough, P. N. 2009. Software Product Management and Pricing: Key Success Factors for Software Organizations. Springer-Verlag New York Inc.

Kittlaus, H.-B., and Fricker, S. 2017. Software Product Management. Berlin, Germany: Springer.

Knauss, E., Lübke, D., and Meyer, S. 2009. "Feedback-Driven Requirements Engineering: The Heuristic Requirements Assistant," in: 31st International Conference on Software Engineering (ICSE 2009). Vancouver, British Columbia, Canada.

Kohavi, R., Rothleder, N. J., and Simoudis, E. 2002. "Emerging Trends in Business Analytics," Communications of the ACM (45:8), pp. 45-48.

Krueger, R. A. 2009. Focus Groups: A Practical Guide for Applied Research. Sage.

Kujala 1, S. 2008. "Effective User Involvement in Product Development by Improving the Analysis of User Needs," Behaviour & Information Technology (27:6), pp. 457-473.

Kujala, S. 2003. "User Involvement: A Review of the Benefits and Challenges," Behaviour & information technology (22:1), pp. 1-16.

Kujala, S., and Miron-Shatz, T. 2013. "Emotions, Experiences and Usability in Real-Life Mobile Phone Use," SIGCHI Conference on Human Factors in Computing Systems, Paris, France: ACM, pp. 1061-1070.

Kurtanović, Z., and Maalej, W. 2018. "On User Rationale in Software Engineering," Requirements Engineering), pp. 1-23.

Kusters, R. J., van Solingen, R., and Trienekens, J. J. 1999. "Identifying Embedded Software Quality: Two Approaches," Quality and Reliability Engineering International (15:6), pp. 485-492.

Le Callet, P., Möller, S., and Perkis, A. 2012. "Qualinet White Paper on Definitions of Quality of Experience," in: European Network on Quality of Experience in Multimedia Systems and Services.

Lee, D. H., and Brusilovsky, P. 2009. "Reinforcing Recommendation Using Implicit Negative Feedback," International conference on user modeling, adaptation, and personalization: Springer, pp. 422-427.

Lehman, M. M. 1980. "Programs, Life Cycles, and Laws of Software Evolution," Proceedings of the IEEE (68:9), pp. 1060-1076.

Lehman, M. M. 1996. "Feedback in the Software Evolution Process," Information and Software technology (38:11), pp. 681-686.

Lehman, M. M., and Ramil, J. F. 2003. "Software Evolution—Background, Theory, Practice," Information Processing Letters (88:1-2), pp. 33-44.

Lehman, M. M., Ramil, J. F., Wernick, P. D., Perry, D. E., and Turski, W. M. 1997. "Metrics and Laws of Software Evolution-the Nineties View," Proceedings Fourth International Software Metrics Symposium: IEEE, pp. 20-32.

Lehrer, K. 2018. Theory of Knowledge. Routledge.

Leucker, M., and Schallhart, C. 2009. "A Brief Account of Runtime Verification," Journal of Logic and Algebraic Programming (78:5), pp. 293-303.

Liu, L., Zhou, Q., Liu, J., and Cao, Z. 2017. "Requirements Cybernetics: Elicitation Based on User Behavioral Data," Journal of Systems and Software (JSS) (124), pp. 187-194.

Lynna, G. S., and Akgünb, A. E. 2001. "Project Visioning: Its Components and Impact on New Product Success," Journal of Product Innovation Management (18:6), pp. 374-387.

Maalej, W., Happel, H.-J., and Rashid, A. 2009. "When Users Become Collaborators: Towards Continuous and Context-Aware User Input," 24th Conference Companion on Object oriented Programming Systems Languages and Applications, Orlando, Florida, USA: ACM, pp. 981-990.

Maalej, W., and Nabil, H. 2015. "Bug Report, Feature Request, or Simply Praise? On Automatically Classifying App Reviews," in: 23rd International Requirements Engineering Conference (RE 2015). Ottawa, Ontario, Canada.

Maalej, W., Nyebi, M., Johann, T., and Ruhe, G. 2016. "Toward Data-Driven Requirements Engineering," IEEE Software (33:1), pp. 48-54.

Maalej, W., and Pagano, D. 2011. "On the Socialness of Software," Ninth IEEE International Conference on Dependable, Autonomic and Secure Computing: IEEE, pp. 864-871.

Madhavji, N. H., Fernandez-Ramil, J., and Perry, D. 2006. Software Evolution and Feedback: Theory and Practice. John Wiley & Sons.

Manikas, K., and Hansen, K. M. 2013a. "Reviewing the Health of Software Ecosystems–a Conceptual Framework Proposal," in: International Workshop on Software Ecosystems (IWSECO 2013). Potsdam, Germany: pp. 33-44.

Manikas, K., and Hansen, K. M. 2013b. "Software Ecosystems–a Systematic Literature Review," Journal of Systems and Software (86:5), pp. 1294-1306.

Marilly, E., Martinot, O., Papini, H., and Goderis, D. 2002. "Service Level Agreements: A Main Challenge for Next Generation Networks," 2nd European Conference on Universal Multiservice Networks (ECUMN 2002) Colmar, France: IEEE, pp. 297-304.

Mattos, D. I., Dmitriev, P., Fabijan, A., Bosch, J., and Olsson, H. H. 2018. "An Activity and Metric Model for Online Controlled Experiments," International Conference on Product-Focused Software Process Improvement: Springer, pp. 182-198.

Maule, A. J., Hockey, G. R. J., and Bdzola, L. 2000. "Effects of Time-Pressure on Decision-Making under Uncertainty: Changes in Affective State and Information Processing Strategy," Acta psychologica (104:3), pp. 283-301.

Menasce, D. A. 2002. "Qos Issues in Web Services," Internet Computing, IEEE (6:6), pp. 72-75.

Milne, A., and Maiden, N. 2012. "Power and Politics in Requirements Engineering: Embracing the Dark Side?," Requirements Engineering (17:2), pp. 83-98.

Minhas, T. N., and Fiedler, M. 2013. "Quality of Experience Hourglass Model," in: International Conference on Computing, Management and Telecommunications (ComManTel). Ho Chi Minh City, Vietnam.

Mitra, K., Zaslavsky, A., and Åhlund, C. 2011. "A Probabilistic Context-Aware Approach for Quality of Experience Measurement in Pervasive Systems," 26th ACM symposium on applied computing, Taichung, Taiwan ACM, pp. 419-424.

Morales-Ramirez, I., Perini, A., and Guizzardi, R. S. 2015. "An Ontology of Online User Feedback in Software Engineering," Applied Ontology (10:3-4), pp. 297-330.

Nadal, S., Herrero, V., Romero, O., Abelló, A., Franch, X., Vansummeren, S., and Valerio, D. 2017. "A Software Reference Architecture for Semantic-Aware Big Data Systems," Information and software technology (90), pp. 75-92.

Nadal, S., Romero, O., Abelló, A., Vassiliadis, P., and Vansummeren, S. 2019. "An Integration-Oriented Ontology to Govern Evolution in Big Data Ecosystems," Information systems (79), pp. 3-19.

Nakhimovsky, Y., Miller, A. T., Dimopoulos, T., and Siliski, M. 2010. "Behind the Scenes of Google Maps Navigation: Enabling Actionable User Feedback at Scale," in Chi'10 Extended Abstracts on Human Factors in Computing Systems. pp. 3763-3768.

Nonaka, I., and Takeuchi, H. 1991. The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation. Oxford university press.

Nonaka, I., and Toyama, R. 2003. "The Knowledge-Creating Theory Revisited: Knowledge Creation as a Synthesizing Process," in The Essentials of Knowledge Management. Springer, pp. 95-110.

Nonaka, I., and Toyama, R. 2015. "The Knowledge-Creating Theory Revisited: Knowledge Creation as a Synthesizing Process," in The Essentials of Knowledge Management. Springer, pp. 95-110.

Oh, J., Lee, S., and Lee, U. 2016. "How to Report App Feedback? Analyzing Feedback Reporting Behavior," Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems, pp. 3173-3180.

Olsson, H. H., Alahyari, H., and Bosch, J. 2012. "Climbing the" Stairway to Heaven"--a Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development Towards Continuous Deployment of Software," 38th euromicro conference on software engineering and advanced applications, Cesme, Izmir, Turkey: IEEE, pp. 392-399.

Olsson, H. H., and Bosch, J. 2014. "Climbing the “Stairway to Heaven”: Evolving from Agile Development to Continuous Deployment of Software," in Continuous Software Engineering. Springer, pp. 15-27.

Oppenheimer, D. M., Meyvis, T., and Davidenko, N. 2009. "Instructional Manipulation Checks: Detecting Satisficing to Increase Statistical Power," Journal of experimental social psychology (45:4), pp. 867-872.

Oriol, M., Stade, M., Fotrousi, F., Nadal, S., Varga, J., Seyff, N., Abello, A., Franch, X., Marco, J., and Schmidt, O. 2018. "Fame: Supporting Continuous Requirements Elicitation by Combining User Feedback and Monitoring," in: 26th IEEE International Conference on Requirements Engineering (RE). Banff, Canada: IEEE, pp. 207-217.

Osterwalder, A., Pigneur, Y., Bernarda, G., and Smith, A. 2014. Value Proposition Design: How to Create Products and Services Customers Want. John Wiley & Sons.

Ottum, B. D., and Moore, W. L. 1997. "The Role of Market Information in New Product Success/Failure," Journal of Product Innovation Management (14:4), pp. 258-273.

Pagano, D., and Brügge, B. 2013. "User Involvement in Software Evolution Practice: A Case Study," 35th international conference on Software engineering (ICSE 2013), San Francisco, CA, USA: IEEE Press, pp. 953-962.

Pagano, D., and Maalej, W. 2013. "User Feedback in the Appstore: An Empirical Study," 21st IEEE international conference on requirements engineering (RE): IEEE, pp. 125-134.

Parmenter, D. 2010. Key Performance Indicators (Kpi): Developing, Implementing, and Using Winning Kpis. John Wiley & Sons.

Peffers, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. 2007. "A Design Science Research Methodology for Information Systems Research," Journal of management information systems (24:3), pp. 45-77.

Petersen, K., Feldt, R., Mujtaba, S., and Mattsson, M. 2008. "Systematic Mapping Studies in Software Engineering," in: 12th International Conference on Evaluation and Assessment in Software Engineering. p. 1.

Phaal, R., Farrukh, C. J., and Probert, D. R. 2004. "Technology Roadmapping—a Planning Framework for Evolution and Revolution," Technological forecasting and social change (71:1), pp. 5-26.

Phaal, R., Farrukh, C. J., and Probert, D. R. 2007. "Strategic Roadmapping: A Workshop-Based Approach for Identifying and Exploring Innovation Issues and Opportunities," Engineering Management Journal (19), pp. 3-12.

Pohl, K., and Rupp, C. 2011. Requirements Engineering Fundamentals: A Study Guide for the Certified Professional for Requirements Engineering Exam - Foundation Level - Ireb Compliant. Rocky Nook Computing.

Potter, W. J., and Levine-Donnerstein, D. 1999. "Rethinking Validity and Reliability in Content Analysis," Journal of Applied Communication Research (27:3), pp. 258-284.

Potts, C., Takahashi, K., and Antón, A. I. 1994. "Inquiry-Based Requirements Analysis," IEEE software (11:2), pp. 21-32.

Qian, W., Peng, X., Wang, H., Mylopoulos, J., Zheng, J., and Zhao, W. 2018. "Mobigoal: Flexible Achievement of Personal Goals for Mobile Users," IEEE Transactions on Services Computing (11:2), pp. 384-398.

Raake, A., and Egger, S. 2014. "Quality and Quality of Experience," in Quality of Experience. Springer, pp. 11-33.

Rabiser, R., Guinea, S., Vierhauser, M., Baresi, L., and Grünbacher, P. 2017. "A Comparison Framework for Runtime Monitoring Approaches," Journal of Systems and Software (125), pp. 309-321.

Rajlich, V. c. T., and Bennett, K. H. 2000. "A Staged Model for the Software Life Cycle," Computer (33:7), pp. 66-71.

Rapport, D. J., Costanza, R., and McMichael, A. J. 1998. "Assessing Ecosystem Health," Trends in Ecology & Evolution (13:10), pp. 397-402.

Rashid, A., Wiesenberger, J., Meder, D., and Baumann, J. 2009. "Bringing Developers and Users Closer Together: The Open Proposal Story," PRIMIUM-Process Innovation for Enterprise Software pp. 9-26.

Regnell, B., Berntsson Svensson, R., and Olsson, S. 2008. "Supporting Roadmapping of Quality Requirements," IEEE Software (25:2), pp. 42-47.

Reiter, U., Brunnström, K., De Moor, K., Larabi, M.-C., Pereira, M., Pinheiro, A., You, J., and Zgank, A. 2014. "Factors Influencing Quality of Experience," in Quality of Experience. Springer, pp. 55-72.

Rettig, M. 1994. "Prototyping for Tiny Fingers," Communications of the ACM (37:4), pp. 21-27.

Robert, L. P., and Dennis, A. R. 2005. "Paradox of Richness: A Cognitive Model of Media Choice," IEEE transactions on professional communication (48:1), pp. 10-21.

Robinson, W. 2009. "A Roadmap for Comprehensive Requirements Modeling," Computer (43:5), pp. 64-72.

Robson, C. 2002. Real World Research: A Resource for Social Scientists and Practitioner-Researchers. Blackwell Oxford.

Roto, V., Law, E., Vermeeren, A., and Hoonhout, J. 2011. "User Experience White Paper- Bringing Clarity to the Concept of User Experience. Results from Dagstuhl Seminar on Demarcating User Experience.," Schloss Dagstuhl, Leibniz-Zentrum for Informatik, Germany.

Salehie, M., and Tahvildari, L. 2012. "Towards a Goal-Driven Approach to Action Selection in Self-Adaptive Software," Software: Practice and Experience (42:2), pp. 211-233.

Sanchez, L., Muñoz, L., Galache, J. A., Sotres, P., Santana, J. R., Gutierrez, V., Ramdhany, R., Gluhak, A., Krco, S., and Theodoridis, E. 2014. "Smartsantander: Iot Experimentation over a Smart City Testbed," Computer Networks (61), pp. 217-238.

Santos, R., Werner, C. u., Barbosa, O., and Alves, C. 2012. "Software Ecosystems: Trends and Impacts on Software Engineering," in: 26th Brazilian Symposium in Software Engineering (SBES 2012). IEEE, pp. 206-210.

Sauerwein, E., Bailom, F., Matzler, K., and Hinterhuber, H. H. 1996. "The Kano Model: How to Delight Your Customers," International Working Seminar on Production Economics, Igls, Innsbruck, Austria, pp. 313-327.

Scherer, K. R. 2005. "What Are Emotions? And How Can They Be Measured?," Social science information (44:4), pp. 695-729.

Schleicher, R., Westermann, T., and Reichmuth, R. 2014. "Mobile Human–Computer Interaction," in Quality of Experience. Springer, pp. 339-349.

Schmitz, J., and Fulk, J. 1991. "Organizational Colleagues, Media Richness, and Electronic Mail: A Test of the Social Influence Model of Technology Use," Communication research (18:4), pp. 487-523.

Schneider, K. 2011. "Focusing Spontaneous Feedback to Support System Evolution," 19th International Conference on Requirements Engineering (RE), Trento, Italy: IEEE, pp. 165-174.

Seyff, N., Ollmann, G., and Bortenschlager, M. 2011. "Irequire: Gathering End-User Requirements for New Apps," 19th IEEE International Requirements Engineering Conference (RE'11), Trento, Italy: IEEE.

Seyff, N., Ollmann, G., and Bortenschlager, M. 2014. "Appecho: A User-Driven, in Situ Feedback Approach for Mobile Platforms and Applications," 1st International Conference on Mobile Software Engineering and Systems (MOBILESoft), Hyderabad, India: ACM, pp. 99-108.

Seyff, N., Stade, M., Fotrousi, F., Glinz, M., Guzman, E., Kolpondinos-Huber, M., Arzapalo, D. M., Oriol, M., and Schaniel, R. 2017. "End-User Driven Feedback Prioritization,").

Seyff, N., Todoran, I., Caluser, K., Singer, L., and Glinz, M. 2015. "Using Popular Social Network Sites to Support Requirements Elicitation, Prioritization and Negotiation," Journal of Internet Services and Applications (JISA) (6:1), pp. 7:1–7:16.

Shaikh, J., Fiedler, M., and Collange, D. 2010. "Quality of Experience from User and Network Perspectives," annals of telecommunications-annales des telecommunications (65:1-2), pp. 47-57.

Shung, K. P., and Junyu, M. C. 2012. "Application of Analytics in Business Strategy," Business Intelligence Journal (5:1).

Sjøberg, D. I., Anda, B., Arisholm, E., Dybå, T., Jørgensen, M., Karahasanović, A., and Vokáč, M. 2003. "Challenges and Recommendations When Increasing the Realism of Controlled Software Engineering Experiments," in Empirical Methods and Studies in Software Engineering, R. Conradi and A.I. Wang (eds.). Berlin Heidelberg: Springer, pp. 24-38.

Snijders, R., Dalpiaz, F., Hosseini, M., Shahri, A., and Ali, R. 2014. "Crowd-Centric Requirements Engineering," 7th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2014), London, UK: IEEE.

Solomon, R. C. 2008. "The Philosophy of Emotions," in Handbook of Emotions, M. Lewis, Haviland-Jones, J. M. & Barrett, L. F. (ed.). New York: Guilford Press, pp. 3-16.

Sotres, P., de la Torre, C. L., Sánchez, L., Jeong, S., and Kim, J. 2018. "Smart City Services over a Global Interoperable Internet-of-Things System: The Smart Parking Case," 2018 Global Internet of Things Summit (GIoTS): IEEE, pp. 1-6.

Souag, A., Salinesi, C., and Wattiau, I. 2012. "Ontologies for Security Requirements: A Literature Survey and Classification," in: Advanced Information Systems Engineering Workshops. Gdańsk, Poland.

Srewuttanapitikul, K., and Muengchaisri, P. 2016. "Prioritizing Software Maintenance Plan by Analyzing User Feedback," 2016 International Conference on Information Science and Security (ICISS): IEEE, pp. 1-5.

Srivastava, J., Cooley, R., Deshpande, M., and Tan, P.-N. 2000. "Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data," ACM SIGKDD Explorations Newsletter (1:2), pp. 12-23.

Stade, M., Fotrousi, F., Seyff, N., and Albrecht, O. 2017. "Feedback Gathering from an Industrial Point of View," 25th International Conference on Requirements Engineering (RE), Lisbon, Portugal: IEEE, pp. 71-79.

Stade, M., and Seyff, N. 2017. "Features for Mobile Feedback Tools: Applying the Kano Method," Mensch und Computer 2017-Tagungsband).

Strohmeier, D., Egger, S., Raake, A., Hoßfeld, T., and Schatz, R. 2014. "Web Browsing," in Quality of Experience. Springer, pp. 329-338.

Suh, K. S. 1999. "Impact of Communication Medium on Task Performance and Satisfaction: An Examination of Media-Richness Theory," Information & Management (35:5), pp. 295-312.

Sun, P.-C., and Cheng, H. K. 2007. "The Design of Instructional Multimedia in E-Learning: A Media Richness Theory-Based Approach," Computers & education (49:3), pp. 662-676.

Sutcliffe, A., and Sawyer, P. 2013. "Requirements Elicitation: Towards the Unknown Unknowns," 2013 21st IEEE International Requirements Engineering Conference (RE): IEEE, pp. 92-104.

Svahnberg, M., Gorschek, T., Feldt, R., Torkar, R., Saleem, S. B., and Shafique, M. U. 2010. "A Systematic Review on Strategic Release Planning Models," Information and software technology (52:3), pp. 237-248.

Szajna, B., and Scamell, R. W. 1993. "The Effects of Information System User Expectations on Their Performance and Perceptions," Mis Quarterly (17:4), pp. 493-516.

Taentzer, G., Goedicke, M., Paech, B., Schneider, K., Schürr, A., and Vogel-Heuser, B. 2019. "The Nature of Software Evolution," in Managed Software Evolution. Springer, pp. 9-20.

Thew, S., and Sutcliffe, A. 2018. "Value-Based Requirements Engineering: Method and Experience," Requirements engineering (23:4), pp. 443-464.

Tohidi, M., Buxton, W., Baecker, R., and Sellen, A. 2006. "User Sketches: A Quick, Inexpensive, and Effective Way to Elicit More Reflective User Feedback," 4th Nordic conference on Human-computer interaction: changing roles, Oslo, Norway: ACM, pp. 105-114.

Turner, M., Kitchenham, B., Brereton, P., Charters, S., and Budgen, D. 2010. "Does the Technology Acceptance Model Predict Actual Use? A Systematic Literature Review," Information and software technology (52:5), pp. 463-479.

Ulriksen, M. S., and Dadalauri, N. 2016. "Single Case Studies and Theory-Testing: The Knots and Dots of the Process-Tracing Method," International Journal of Social Research Methodology (19:2), pp. 223-239.

Van der Ham, W. F., Broekens, J., and Roelofsma, P. H. 2014. "The Effect of Dominance Manipulation on the Perception and Believability of an Emotional Expression," in Emotion Modeling: Towards Pragmatic Computational Models of Affective Processes, T. Bosse, Broekens, J., Dias J., & Zwaan, J. v. d (ed.). Springer, pp. 101–114.

van Hoorn, A., Rohr, M., Hasselbring, W., Waller, J., Ehlers, J., Frey, S., and Kieselhorst, D. 2009. Continuous Monitoring of Software Services: Design and Application of the Kieker Framework. Department of Computer Science, Kiel University, Germany.

Varela, M., Skorin-Kapov, L., and Ebrahimi, T. 2014. "Quality of Service Versus Quality of Experience," in Quality of Experience. Springer, pp. 85-96.

Vierhauser, M., Rabiser, R., and Grünbacher, P. 2016. "Requirements Monitoring Frameworks: A Systematic Review," Information and Software Technology (80), pp. 89-109.

Wang, T., Si, Y., Xuan, X., Wang, X., Yang, X., Li, S., and Kavs, A. J. 2010. "A Qos Ontology Cooperated with Feature Models for Non-

Functional Requirements Elicitation," Proceedings of the Second Asia-Pacific Symposium on Internetware, Suzhou, China: ACM, p. 17.

Wang, Y., Mcilraith, S. A., Yu, Y., and Mylopoulos, J. 2009. "Monitoring and Diagnosing Software Requirements," Automated Software Engineering (16:1), p. 3.

Watson-Manheim, M. B., and Bélanger, F. 2007. "Communication Media Repertoires: Dealing with the Multiplicity of Media Choices," MIS quarterly), pp. 267-293.

Wehrmaker, T., Gärtner, S., and Schneider, K. 2012. "Contexter Feedback System," 2012 34th International Conference on Software Engineering (ICSE): IEEE, pp. 1459-1460.

Weiblen, T., Giessmann, A., Bonakdar, A., and Eisert, U. 2012. "Leveraging the Software Ecosystem-Towards a Business Model Framework for Marketplaces," in: Dcnet/ice-b/optics. pp. 187-193.

Wellsandt, S., Hribernik, K., and Thoben, K. 2014. "Qualitative Comparison of Requirements Elicitation Techniques That Are Used to Collect Feedback Information About Product Use," in: 24th CIRP Design Conference. Milano, Italy.

Westin, S. S. 1998. "Performance Measurement and Evaluation: Definitions and Relationships," GAO/GGD-98-26. Washington, DC: US Government Printing Office.

Wieringa, R., Maiden, N., Mead, N., and Rolland, C. 2006. "Requirements Engineering Paper Classification and Evaluation Criteria: A Proposal and a Discussion," Requirements Engineering (11:1), pp. 102-107.

Wieringa, R. J. 2014. Design Science Methodology for Information Systems and Software Engineering. Springer.

Wüest, D., Fotrousi, F., and Fricker, S. 2019. "Combining Monitoring and Autonomous Feedback Requests to Elicit Actionable Knowledge of System Use," Cham: Springer International Publishing, pp. 209-225.

Wüest, D., Seyff, N., and Glinz, M. 2012. "Flexisketch: A Mobile Sketching Tool for Software Modeling," International Conference on Mobile Computing, Applications, and Services: Springer, pp. 225-244.

Wüest, D., Seyff, N., and Glinz, M. 2015. "Sketching and Notation Creation with Flexisketch Team: Evaluating a New Means for Collaborative Requirements Elicitation," 23rd IEEE International Requirements Engineering Conference (RE'15), Ottawa, Canada: IEEE, pp. 186-195.

XES_WorkingGroup. 2016. "Ieee Standard for Extensible Event Stream (Xes) for Achieving Interoperability in Event Logs and Event Streams," in: IEEE Std 1849.

Yetim, F., Draxler, S., Stevens, G., and Wulf, V. 2012. "Fostering Continuous User Participation by Embedding a Communication Support Tool in User Interfaces," AIS Transactions on Human-Computer Interaction (4:2), pp. 153-168.

Yin, R. K. 2014. Case Study Research: Design and Methods, (5 ed.). Thousands of Oaks, CA: Sage publications.

Yusop, N. S. M., Grundy, J., and Vasa, R. 2015. "Reporting Usability Defects: Limitations of Open Source Defect Repositories and Suggestions for Improvement," Proceedings of the ASWEC 2015 24th Australasian Software Engineering Conference, pp. 38-43.

Zhang, D., Dang, Y., Lou, J.-G., Han, S., Zhang, H., and Xie, T. 2011. "Software Analytics as a Learning Case in Practice: Approaches and Experiences," in: International Workshop on Machine Learning Technologies in Software Engineering. Lawrence, Kansas, U.S.A. : pp. 55-58.

Zhang, J., and Ansari, N. 2011. "On Assuring End-to-End Qoe in Next Generation Networks: Challenges and a Possible Solution," IEEE Communications Magazine (49:7), pp. 185-191.

Zheng, X., Julien, C., Podorozhny, R., Cassez, F., and Rakotoarivelo, T. 2016. "Efficient and Scalable Runtime Monitoring for Cyber–

Physical System," IEEE Systems Journal (12:2), pp. 1667-1678.

Zijlstra, F. R., Roe, R. A., Leonora, A. B., and Krediet, I. 1999. "Temporal Factors in Mental Work: Effects of Interrupted Activities," Journal of Occupational and Organizational Psychology (72:2), pp. 163-185.

Zimmermann, T., Premraj, R., Bettenburg, N., Just, S., Schroter, A., and Weiss, C. 2010. "What Makes a Good Bug Report?," IEEE Transactions on Software Engineering (36:5), pp. 618-643.

Zowghi, D., and Coulin, C. 2005. "Requirements Elicitation: A Survey of Techniques, Approaches, and Tools," in Engineering and Managing Software Requirements, A. Aurum and C. Wohlin (eds.). Berlin, Germany: Springer.

Zowghi, D., Da Rimini, F., and Bano, M. 2015. "Problems and Challenges of User Involvement in Software Development: An Empirical Study," 19th International Conference on Evaluation and Assessment in Software Engineering (EASE), Nanjing, China: ACM, pp. 1-10.

ABSTRACT

Context: Companies continuously explore their software systems to acquire evidence for software evolution, such as bugs in the system and new functional or quality requirements. So far, managers have made decisions about software evolution based on evidence gathered from interpreting user feedback and monitoring data collected separately from software in use. These evidence-collection processes are usually unmethodical, lack a systematic guide, and have practical issues. This lack of a systematic approach leaves unexploited opportunities for detecting evidence for system evolution.

Objective: The main research objective is to improve evidence collection from software in use and guide software practitioners in decision-making about system evolution. Understanding useful approaches to collect user feedback and monitoring data, two important sources of evidence, and combining them are key objectives as well.

Method: We proposed a method for gathering evidence from software in use (GESU) using design-science research. We designed the method over three iterations and validated it in the European case studies FI-Start, Supersede, and Wise-IoT. To acquire knowledge for the design, we conducted further research using surveys and systematic mapping methods.

Results: The results show that GESU is not only successful in industrial environments but also yields new evidence for software evolution by bringing user feedback and monitoring data together. This combination helps software practitioners improve their understanding of end-user needs and system drawbacks, ultimately supporting continuous requirements elicitation and product evolution. GESU suggests monitoring a software system based on its goals to filter relevant data (i.e., goal-driven monitoring) and gathering user feedback when the system requests feedback about the software in use (i.e., system-triggered user feedback). The system identifies interesting situations of system use and issues automated requests for user feedback to interpret the evidence from user perspectives. We justified using goal-driven monitoring and system-triggered user feedback with complementary findings of the thesis. That showed the goals and characteristics of software systems constrain monitoring data. We thus narrowed the monitoring and observational focus on data aligned with goals instead of a massive amount of potentially useless data. Finally, we found that requesting feedback from users with a simple feedback form is a useful approach for motivating users to provide feedback.

Conclusion: Combining user feedback and monitoring data is helpful to acquire insights into the success of a software system and guide decision-making regarding its evolution. This work can be extended in the future by implementing an adaptive system for gathering evidence from combined monitoring data and user feedback.

2020:04

ISSN: 1653-2090
ISBN: 978-91-7295-402-1