# PORTNEUF – A Framework for Continuous User Involvement

Dennis Pagano

INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Forschungs- und Lehreinheit I
Angewandte Softwaretechnik

# PORTNEUF – A Framework for Continuous User Involvement

Dennis Pagano

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

**Doktors der Naturwissenschaften (Dr. rer. nat.)**

genehmigten Dissertation.

Die Dissertation wurde am 30.01.2013 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 25.05.2013 angenommen.

*To Anna with love.*

# Acknowledgements

This work would not have been possible without the support of many other people. I would like to take this opportunity to thank all of you, but as a famous researcher once wrote, “there is not enough space in this edge”. If you do not read your name here, be sure it is memorized in my heart.

First, I would like to express my deep gratitude to Bernd Bruegge, who has been much more to me than a supervisor. He was an infinite source of inspiration, passion, and enthusiasm, and really never let me down. With his trust in me and in my work, he gave me the room to become an independent and responsible researcher, engineer, and person. Further, I want to thank Kurt Schneider, whose research inspired me to study software users. His spirit of research was a great example to me and his feedback always opened up new perspectives.

I am very grateful to all members of the Chair for Applied Software Engineering. I really learned a lot from all of you, and am indebted for your support and the social environment you all have contributed to. Three of my friends and colleagues influenced me in particular. Damir Ismailović, who has always been there for me, day and night, and whose pragmatic way and friendship was essential for me to follow through. Walid Maalej, who has been an extremely professional and motivating guide throughout my research. I have benefited very much from his experience and from our collaboration in numerous occasions. Tobias Roehm, who was a great room mate and a true believer in science, and whose well structured and positive way often helped me to find my balance.

I am further indebted to Oliver Neumann, who piqued my interest in mathematics and computer science, allowing me to join his mathematical journeys and expeditions, to Wolfgang Groh, who opened my eyes and taught me how important it is to look behind the scenes of our society and to scrutinize the apparent, to Anne Brüggemann-Klein, who gave me the opportunity to start my scientific career, and to Florian Rhöse, my spiritual guide and brother.

I thank my students, co-authors, and all other persons who helped me, in particular Rana Alkadhi, Daniel Bader, Krisna Haryanto, Andreas Hegenberg, Felix Kaser, Amel Mahmuzić, Yichen Mao, Gerhard Miller, Helmut Naughton, Martin Ott, Patrick Renner, Christoph Teschner, and Christian Ziegler.

Finally, I want to express my love and gratitude to my family, and in particular to my wife Anna. Writing a dissertation requires even more than the researcher’s full attention. I am indebted for your love and devotion, your constant support and understanding. You are the source of joy and happiness in my life.

# Abstract

Post-deployment user feedback such as feature requests and bug reports have the potential to improve software quality and reveal missing functionality. Nevertheless, in practice software companies often ignore user feedback and simply stick with predefined roadmaps and development plans, because of two main problems. First, in order to benefit from user feedback, developers have to analyze, consolidate, and structure it, which becomes time-consuming when feedback volume is high. Second, in order to prioritize their work, developers need to assess how many users are affected by a particular feedback, by identifying duplicate and similar feedback in a manual way.

This thesis describes PORTNEUF, a framework which tackles both problems. It consolidates user feedback by employing a context-aware, hybrid recommender system. Moreover, it introduces FEEDBACKRANK, an algorithm which allows developers to assess the importance of collected user feedback to the user community.

We demonstrate the feasibility and applicability of our approach in two real-world applications. In a quasi-experiment with a large number of users, we showed that PORTNEUF increases developers' efficiency when working with user feedback. The framework significantly reduces the amount of user feedback duplicates by over 67%, with the recommender system showing a hit-rate of over 82%. Moreover, we obtained an average precision of around 76% for FEEDBACKRANK, which suggests that it provides a valuable estimation of what is important to the user community.

# Contents

# Chapter 1

# Introduction

*«[...] pay attention to what users do, not what they say.»*

— Jakob Nielsen

Software engineering aims at developing *useful* software systems. To this end, software engineers often elicit needs and expectations of prospective users and formalize them into requirements which drive further development efforts. But once deployed, the developed system must *evolve* with changing end user needs to remain useful and relevant [49]. However, evolutionary software changes and enhancements might also affect a system's usefulness, if they are not in line with users' expectations.

The goal of *user involvement* is to improve the usefulness of a system by obtaining a thorough understanding of user needs and expectations. Traditionally, user involvement referred to the actual participation of selected, representative users in development activities. But over the last decades, both hardware and software systems underwent fundamental changes, and as a result also the software *users* changed – from few engineers with special scientific requirements to large amounts of "ordinary" people with various needs. Moreover, software systems have become part of our everyday life, and the mobile and ubiquitous devices we interact with have added an additional layer of diversity to the various application scenarios that software developers have to deal with. As a consequence, user involvement – like software development itself – must not end with the delivery of a system, but accompany and safeguard its evolution over time.

Moreover, from a strategic point of view, it was never as important as now for software companies to meet the expectations of their end users. On the one hand, free and open source competitors lurk at every corner, finding more and more supporters and gaining industrial relevance. On the other hand, end users themselves are getting more knowledgeable regarding the tools they want to use, and the quality they ask for. Todays' users grow up with high-speed internet, highly reactive, touch-based, natural user interfaces, context-aware software, and sensor-equipped, mobile devices.

In addition, the internet and specifically social media have accelerated word-of-mouth recommendations by orders of magnitude. As a consequence, frustrated users meet in social communities to argue against specific software and might even harm its reputation[1]. The example of Final Cut Pro X – a software for professional video editing – nicely illustrates the emerging effects:

On June 21, 2011, Apple released a new version of the software Final Cut Pro, after having made some revolutionary changes. In particular, several core features were removed, the user interface was modified in a way that affected standard workflows, and the software was incompatible with previous versions so that existing filmmaking projects could not be imported [28]. Filmmakers, who suspected that Apple wanted to shift their focus to a more consumer oriented market, reacted in a devastating way. They created an online petition titled "Final Cut Pro X is Not a Professional Application" [169], asking Apple amongst others to immediately reinstate and support the previous version. News spread like wildfire and after one week over 600 professional filmmakers had signed the petition [93]. Apple's first reaction was to ban the initiator's Apple account and to shut down without notice the corresponding discussion threads in the Apple support forum [178]. Was that the right thing to do? Was that Jakob Nielsen's intention when he said "pay attention to what users do, not what they say" [207]?

## 1.1 Problem Statement

Hardware, software, users, and communication channels between users and developers have changed significantly over the last decades, leading to two major consequences [182]. First, the importance of user involvement has increased. Larger amounts of demanding users, software which is used in changing mobile contexts, and a stronger competition, call for an increased focus on users. Second, user involvement has to accompany software development *continuously*, i.e. throughout the whole software lifecycle. As a consequence, post-deployment *user feedback* such as feature requests and bug reports should become increasingly important to developers [162].

In practice, however, resource constraints often do not allow companies to employ effective user involvement methods [245]. In addition, there are a number of communication gaps between users and developers, which affect the efficiency of user involvement [121, 136, 181].

Specifically, when working with user feedback, developers experience the following two problems:

1. **User feedback is intractable when its volume is high**. Developers need to *consolidate*, *structure*, *analyze*, and *track* user feedback, in order

[1]For instance http://ihatelotusnotes.com or http://dreckstool.de/hitlist.do

to be able to react on it. However, user feedback is typically written in natural language, might have poor quality, and could reflect contradicting user preferences, which leads to considerable effort for user involvement, in particular when feedback volume is high.

2. **User feedback impact is latent**. In order to *prioritize* their work, developers need to *estimate* how many users experience a specific reported error or desire a particular requested feature. To this end, they have to manually identify and group duplicate or similar feedback which is often scattered across several communication channels. Therefore, great efforts are necessary to assess user feedback impact.

As a result of these problems, software development companies often ignore feedback and simply stick with their predefined product roadmaps and development plans.

## 1.2 Contribution

Our thesis proposition is that users should be involved continuously and systematically during software evolution. In contrast to Nielsen, we suggest:

*Pay attention to what users do,* ***and*** *what they say.*

We make three contributions which tackle the identified problems and enhance the body of knowledge on user involvement. First, we provide empirical evidence on the status quo and desired state of user involvement during software evolution. Second, we develop a lightweight framework which facilitates the analysis of user feedback and allows developers to assess its impact. Third, we show the feasibility and efficiency of our approach by evaluating a reference implementation of the proposed framework with a real-world application and a large number of users.

### Status Quo Assessment

To assess the status quo of user involvement in research and in software evolution practice, we explore the following two research questions:

**RQ 1.** *Which approaches have been described in literature for involving end users during software engineering and especially software evolution activities?*

**RQ 2.** *How and why do software companies involve end users during software evolution in practice, and which challenges and problems arise, if a large number of end users continuously provides feedback?*

To address RQ 1, we review existing literature about user involvement in software engineering. To answer RQ 2, we present an empirical case study which we conducted with software professionals.

## Continuous User Involvement Framework

We establish the theoretical foundations of our solution to the identified problems by deriving a grounded theory on continuous user involvement that relies on user communities. To this end, we study two research questions:

**RQ 3.** *How do software users and developers communicate in open source communities and what can we learn from this communication to improve continuous user involvement?*

**RQ 4.** *How do users provide feedback in application distribution platforms and how can this help us to consolidate user feedback automatically?*

To address RQ 3 and RQ 4 we present the results of two large exploratory studies.

We then substantiate the acquired theoretical solution concepts into a context-aware model for the consolidation and prioritization of user feedback. To this end, we investigate the following research question:

**RQ 5.** *How can the software user community be utilized to consolidate and prioritize user feedback in a proactive way? In particular:*

- How can a recommender system for user feedback identify which existing feedback is relevant for a specific user?
- How can user feedback be rated and prioritized according to its value for the user community?

We address RQ 5 by deriving a domain-independent model which allows for the consolidation and prioritization of user feedback. We further show its applicability to three phases of a typical software engineering lifecycle: early design, system testing, and software evolution. Finally, we describe a reference framework which instantiates the model and details our algorithms.

## Implementation and Evaluation

We evaluate the proposed solution by applying a reference implementation of the PORTNEUF framework to a real-world setting, posing the following research question:

**RQ 6.** *Is our approach feasible and does it increase developers' efficiency when working with user feedback? In particular:*

- Can PORTNEUF consolidate user feedback and thus reduce the amount of duplicates?
- Does PORTNEUF allow for an efficient impact assessment of user feedback?

To address RQ 6, we conduct a quasi-experiment and semi-structured interviews with developers.

### Previously Published Material

Parts of the contributions presented in this dissertation have been published in [14, 99, 182, 217, 218, 219, 220, 221, 222].

## 1.3 Scope

User involvement in software engineering is a broad, socio-technical topic. We therefore limit the scope of user involvement for the purpose of this dissertation regarding the three following dimensions.

1. **Software Lifecycle Phases**. We believe that user involvement is relevant throughout the software lifecycle. Most existing literature about user involvement targets the early phases of a typical software lifecycle, in particular requirements elicitation. In contrast, we focus on user involvement during *software evolution*. More generally, we assume that there is an executable system, be it a prototype or software release, on which developers can gather feedback from its users. Even though we restrict ourself to software evolution, we show how to apply our solution to other software lifecycle phases.

2. **Software System Type**. Software ranges from user-written macros over embedded systems to large tool suites for desktops and mobile applications. In this dissertation, we are interested in *interactive software systems*, i.e. software where users are the primary initiators of an interaction with the system. While we concentrate on interactive software systems, we show how our solution could be applied to other software such as embedded systems.

3. **User Community Size**. Software products have various user communities with different interests and expectations. While we do not restrict the application domain of software, we are primarily interested in settings with large numbers of users, where the identified problems unfold their serious effects most demonstratively.

## 1.4 Contents

The remainder of this dissertation is organized as follows.

Chapter 2 introduces the foundations of user involvement, including definitions and methods as well as benefits and challenges. In Chapter 3 we present an empirical case study conducted with software professionals to analyze the current practice of user involvement during software evolution, specifically for software with large user audiences. Chapter 4 establishes the theoretical foundations of our solution by deriving a grounded theory on continuous user involvement which relies on user communities. For this purpose, we did two large exploratory studies. First, we analyzed the communication between users and developers in open source communities. Second, we investigated user feedback in application distribution platforms. In Chapter 5 we substantiate the obtained theoretical solution concepts into PORTNEUF – a context-aware and domain-independent model for the consolidation and prioritization of user feedback. Our solution harnesses the user community to group user feedback in a proactive way, while increasing its overall quality. In addition, we describe a reference framework which instantiates the model and details our algorithms. Chapter 6 reports on the evaluation of the PORTNEUF framework. Finally, in Chapter 7 we conclude the dissertation, summarizing its contributions and limitations, and illustrating research opportunities that cover topics and questions which emerge from our work.

# Chapter 2

# Foundations

«*The notion of 'user' cannot be precisely defined, and therefore has no place in computer science or software engineering.*»

— Edsger W. Dijkstra, 1979

User involvement in product development is nothing new. Long before the age of computers, users of "traditional" systems would complain about their impracticality due to the too narrow point of view of its creators [124]. Their dissatisfaction has even brought users to take the role of inventors, and thus led to new products created by users [246, 281].

The area of concern of this dissertation is user involvement during software evolution, i.e. our goal is to improve released software systems using contributions by its users. In computer science, user involvement is an established research field [165], which has been studied particularly at its intersection with human-computer-interaction [24]. Involving users during any phase of the software lifecycle is a socio-technical issue, leading to research problems and practical challenges. As it turns out, it is particularly difficult to choose the "right" user involvement methodology in practice, and to assess the efficiency of user involvement [146].

This chapter reviews user involvement literature[1]. In Section 2.1, we explore different definitions of user involvement as used by the community. In Section 2.2, we review user involvement methods and summarize their properties. In Section 2.3, we discuss user involvement experiences in research and practice and illustrate the reported effects. Section 2.4 summarizes our conclusions.

[1]To find relevant publications for this literature review, we have utilized the internet search facilities provided by the main research publishers, such as ACM [3], IEEE [141], Springer [266], and Elsevier [94]. Subsequently, we recursively analyzed and retrieved references from the most relevant papers.

## 2.1 User Involvement Definitions

Although *user involvement* is an established research field [135, 165], there is no commonly accepted definition and the term has been used synonymously with other terms.

Ives and Olson [146] define user involvement as “participation in the system development process by representatives of the target user group”. Two aspects of this definition are noteworthy. First, the notion of a “representative” anticipates a decision *whom*, i.e. which users, to involve. Second, the authors refer to the system “development” process, and thereby limit the *time* of involvement, in particular they exclude the time after delivery.

Wilson et al. [288] take a quite general perspective, defining user involvement as a “focus on users” and emphasizing the “role of users” during system design. On closer inspection, the authors study an in-house development project where users were involved in different ways, being interviewed and observed, as well as evaluating prototypes and the final product.

Heinbokel et al. [135] use the term “user participation” to refer to the involvement of users in design teams, which they regard as a form of “user centeredness” in software engineering. Heiskari and Lehtola [136] object this and claim that user participation should instead be considered only as one form of user involvement. The authors thereby confirm the view of Damodaran [74] that participation denotes one specific level of user involvement.

Grudin [120] highlights gaps between users and developers and describe user involvement as engineers’ “contact with system users”. Maalej et al. [181] support this perspective and refer to user involvement as “gathering and understanding user input”. The authors extend Grudin’s view introducing *communication* and *feedback* as two dimensions of user involvement.

Rashid et al. [235, 236] add the term “user integration” to the list of synonyms used in the community. As it turns out, this term is rarely also used in more practical environments (e.g. [57]) to denote that users are made part of an already existing or thought-out development process.

Barki and Hartwick [17, 18] claim that the term user involvement is misleading and “should be used to refer to a subjective psychological state of the individual reflecting the importance and personal relevance that IS users attach to a given system”, as in other research disciplines. Instead, the term “user participation” should be used “when referring to the various design-related behaviors and activities that the target users or their representatives perform during the system development process”. Despite the quite profound argumentation by the authors, they did not succeed in convincing the community.

In a meta-study, Kujala [165] reviews several definitions and suggests to define user involvement as “a general term describing direct contact with users and covering many approaches”, due to the many different perspectives present in the community. Although this statement has been cited in several other works, we argue that it lacks preciseness and therefore should not be regarded a definition. Rather, the author has given as general and abstract a description that all existing views and definitions would still fit in.

There have been numerous, albeit tentative, attempts to define user involvement over the last decades. But at the same time, hardware and software systems, as well as software engineering have undergone fundamental changes, leading to completely different attitudes towards users and their potential value, and thus eventually to different notions of "user involvement". When reviewing this literature, the accurate researcher needs to go beyond what is written and interpret it in the context of the time.

Until around 1975 software users were mainly engineers or programmers, so there was no need for taking special care of involving users. In the early 1980s, with the emergence of the personal computer, software became more interactive and its users became ordinary people. Research started to look at users and user satisfaction in a different light [120]. In particular the human-computer interaction community underlined the importance of understanding users' needs and tasks, coining the notion of *usability* [55, 116, 209].

Boehm [41] introduced the spiral model, after discovering that the until then prevalent waterfall model [249] did not provide enough flexibility for "particularly interactive end-user applications". Boehm's spiral model for the first time makes prototyping and iterative design first order citizens in software engineering, which allows developers to react more flexibly on changing user requirements.

With the recent emergence of mobile and ubiquitous computers, software and hardware systems again undergo a fundamental change. Changing usage contexts prevent a precise up-front specification of how the software will be used. Further, with the advent of application distribution platforms such as Apple's App Store [13] any person can become a user of any software, thus making it difficult to assess this population beforehand. It has also recently been shown that small businesses account for more than 75% of the top companies present in these platforms [112]. To be applicable, user involvement will consequently have to respect also their limited possibilities and provide cost-effective, lightweight methods.

**Conclusion 1.** There is no commonly accepted definition of user involvement in research and practice. The concept is *multi-dimensional* and context-dependent. Existing research has not systematically specified and established its dimensions. Different perspectives and practical implementations of user involvement instantiate these dimensions in different ways. Finally, user involvement is *subject to change.* Therefore, an enduring definition needs to be extensible.

**Definition 1. User involvement** denotes a systematic *exchange of information* between (prospective) users and developers, with the common *goal* to max-

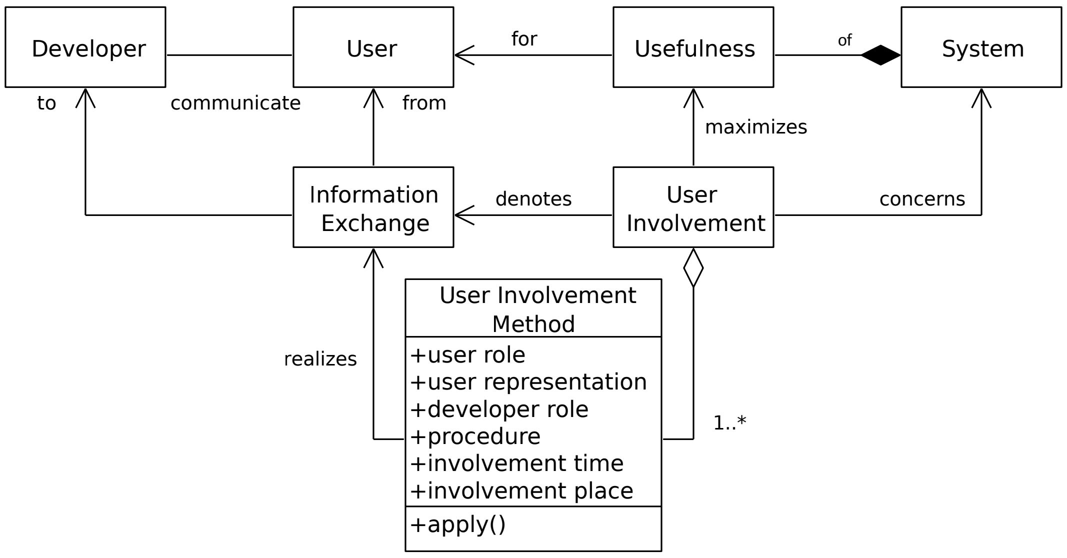


Figure 2.1: User involvement and user involvement methods.

imize system *usefulness* in a specific context. Information exchange is carried out by means of *user involvement method*s.

With this definition, we attempt to capture the two necessary properties of user involvement: its *goal* and the *flow of information* from users to developers. In this regard, we confirm the statement of Heinbokel et al. [135] that "both functionality and usability of software depend on a transfer of knowledge from users to software designers". It is also worth noting that we deliberately separate user involvement from the methods actually used to accomplish it.

**Definition 2.** A **user involvement method** provides for a systematic information exchange between users and developers. It specifies at least the *user role*, the *user representation*, the *developer role*, the *involvement time*, the *involvement place*, and the *procedure* to be followed.

User involvement methods specify how the exchange of information between users and developers takes place. The dimensions in our definition originate from existing literature on user involvement. Bekker and Long [24] analyzed five user involvement approaches and proposed an analysis framework including a total of 19 "attributes", which should enable designers to compare and choose between these approaches. For our definition of user involvement methods, we selected the attributes user role, user representation, developer role, and involvement time, which are related to user-developer communication. Involvement time and place as well as involvement procedure were investigated by Muller et al. [198]. Figure 2.1 illustrates the above definitions. In the following section, we review the literature about user involvement methods and their dimensions.

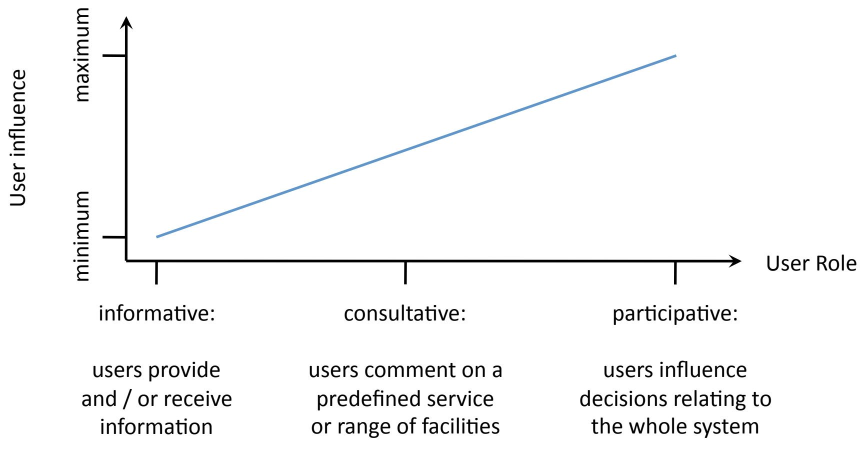

User influence
maximum
minimum
User Role
informative:
users provide and / or receive information
consultative:
users comment on a predefined service or range of facilities
participative:
users influence decisions relating to the whole system

Figure 2.2: User roles and their corresponding influence (adopted from [74]).

## 2.2 User Involvement Methods

In this section, we review existing user involvement methods. There are only a few meta-publications (e.g. [24, 198]) which list different user involvement methods together with their properties. Their goal is to simplify the selection process for researchers and practitioners when looking for a method for a specific purpose or suitable for a given situation. Rather than going through these methods one by one, we investigate what has been published about different attributes and dimensions of user involvement methods.

### 2.2.1 User Role

Bekker and Long [24] distinguish three user roles, *design for*, *design by*, and *design with*, the former two of which had been already introduced by Eason [86]. These roles represent different levels of influence by the user during system design. According to Bekker and Long, the 'design for' user role does not require users to be present during design. The 'design with' user role indicates that users provide "indirect input to the design process". The 'design by' user role expects users "to be directly involved in the design process".

Mumford [202] also describes three roles of users, *consultative*, *representative*, and *consensus*. The consultative user role allows users to influence design decisions, but they are eventually made by the developers. The representative user role involves user groups in the design team. The consensus user role aims at a complete interweaving of users and developers. Mumford originally illustrates that therefore all developers should be involved "in the user department continuously throughout the systems design process" [202] – an advice not necessarily always possible in current software engineering projects.

Table 2.1: Comparison between different user roles.

| | *passive roles* | *active roles* | *decisive roles* |
|---|---|---|---|
| Bekker and Long [24] | design for | design with | design by |
| Mumford [202] | — | consultative, representative | consensus |
| Damodaran [74] | informative | consultative | participative |

Damodaran [74] provides a general view on user roles without any restrictions on physical presence. The author claims that user involvement approaches fall "somewhere on the continuum" from *informative*, through *consultative*, to *participative* [74, 136]. Figure 2.2 illustrates these roles and the corresponding influence of users on the system. The informative user role only indicates an information exchange between users and developers. The consultative user role means that users are provided either with prototypes which they should comment on or with possible design choices from which they should select one. Users filling a participative role may directly influence decisions regarding the system.

To summarize, there are multiple, overlapping user role descriptions in literature. The fact that they all propose three distinct roles is insignificant and does not mean that the roles match. On the contrary, the definitions are expressed using different attributes of the relationship between users and developers. For instance, Bekker and Long [24] refer to the degree of influence, while Damodaran [74] actually specifies the activity carried out by a user filling a role. We agree with the view of Damodaran [74] that user roles can be arranged on a continuum representing increasing influence on design decisions. Table 2.1 compares between the three reviewed models of user roles regarding the type of activity assigned users may perform.

However, there are also user roles which give rise to a suboptimal user-developer communication. Damodaran [74], who provides guidelines for user involvement, points out these common pitfalls. According to the author, users may find themselves in two equally negative roles, the "hostage" role or the "propagandist" role [74], which were originally described by Hedberg [134]. In the *hostage* role, involvement is blocked by the developers. Consequently, users act in a way that promotes "social comfort" but "limits real communication", so that eventually no problems are solved [74]. According to Damodaran, the hostage phenomenon might be caused by inadequate and inappropriate training, what might confuse the users' feelings and lead to a misconception of their role in the project. Moreover, they may perceive software engineering as mysterious, what even reinforces their bias. Damodaran concludes that the hostage role is "particularly damaging because it fails to deliver any of the benefits of user involvement while sustaining the delusion that users are represented in the design team" [74]. The *propagandist* role was originally labeled "indoctrination alternative" by Hedberg [134]. The role denotes the exposure of users to trainings in system design methods, in order to avoid any potential disruptions to design. As it turns out, such training typically changes the perspective of the involved users, and frequently they "begin to adopt the designers' view of the design process" [74]. Damodaran explains that indoctrinated users tend to perceive user problems from a developer perspective and therefore fail to safeguard users' needs.

Interestingly, as Heiskari and Lehtola [136] point out, most user roles do not actually give decisive power to the users. According to Damodaran this is not sufficient, since users need to be able "to influence design, not merely 'rubber stamp' it" [74]. To determine the right degree of user involvement is one of the most discussed issues among user involvement literature. Grudin pinpoints this as follows:

> "What is the optimal degree of user participation in development? If you are developing a compiler, users' involvement will be minimal. If you are copying features from an existing product in a mature application area, limited contact with potential users can be adequate. If you are developing an interactive system in a new domain, full collaboration with users can be essential." [121]

The optimal degree of user involvement thus needs to be decided from case to case. The question remains, however, which range the degree of user involvement may take. Ives and Olson [146] distinguish the following six degrees of user involvement according to the users' influence on the product.

1. *No involvement.* Users are not involved, either because they are not willing or not invited to contribute.

2. *Symbolic involvement.* Users' input is requested, but in fact ignored. While this seems hypocritical, the reason for discarding the gathered information may be manifold. For instance, its quality may be insufficient or the goals not reachable within available budget and time.

3. *Involvement by advice.* Users' advice is requested on a per-issue basis through interviews or questionnaires.

4. *Involvement by weak control.* Users may sign-off at every stage during the development process.

5. *Involvement by doing.* Users are members of the design team or the official liaison to the development team.

6. *Involvement by strong control.* Users may pay themselves for the development effort, or "the user's overall organizational performance evaluation depends on the outcome of the development effort."

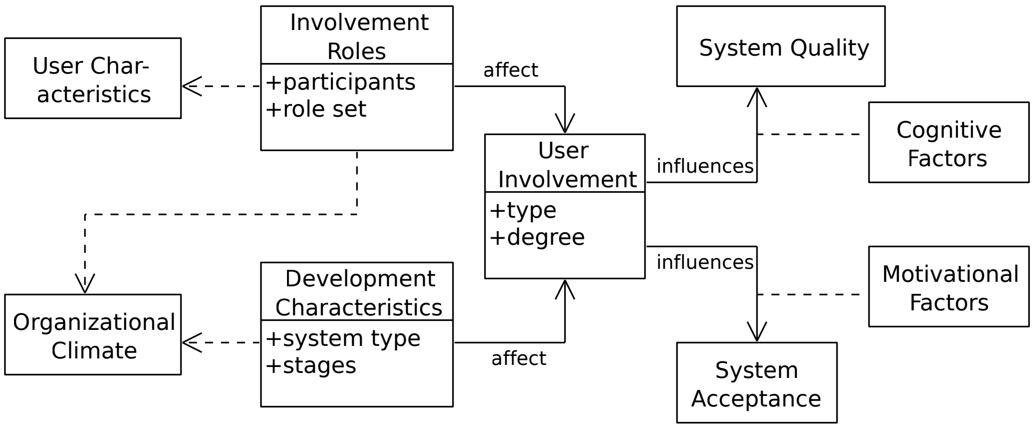


Figure 2.3: Descriptive user involvement model according to Ives and Olson (adopted from [146]).

To summarize, at one extreme developers make assumptions about user requirements while actually ignoring the users themselves, whereas at the other extreme users become designers or collaborators to decide about system acceptance based on self-defined quality criteria.

Ives and Olson further describe which parts of the system users may actually influence. Figure 2.3 illustrates the authors' descriptive model of user involvement, which names system quality and system acceptance as the main "outcome variables" of user involvement [146]. Moreover, the model includes two "intervening mechanisms" between user involvement and its outcome. *Cognitive factors* denote the increased understanding of the system, improved assessment of system needs, and an improved evaluation of the system features. *Motivational factors* on the other hand, denote an increase in the perceived system ownership by the involved users, decreasing resistance to change, and increasing commitment to the new system. While these motivational factors may sound exclusively positive, we argue that these factors might actually push users into the propagandist role.

**Conclusion 2.** User involvement can stand for various degrees of decisive power for the involved users, leading to a continuum from passive to decisive user roles. How much influence is optimal depends on many factors, which is why there is no common agreement in literature. Therefore, researchers have published comparative studies in order to help practitioners to select suitable user involvement methods. In any case, users should be handled with care, since they might easily get blocked as outsiders or change their perspectives when exposed to trainings.

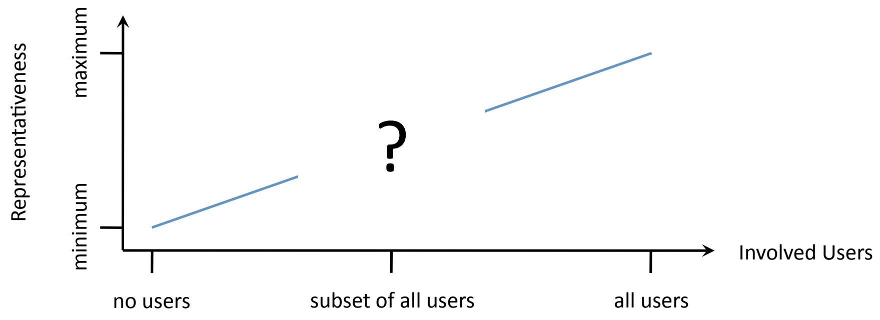


Figure 2.4: Different user representations. The representativeness of a user subset might be unknown.

### 2.2.2 User Representation

User representation refers to the question *who*, i.e. which users, should be involved in the software development process. We need to distinguish two situations. If an existing system is to be improved, i.e. during *software evolution*, actual users are available, while in the case of *greenfield engineering* projects [49], only prospective users may be involved. In either case it is crucial to involve the "right" users.

Damodaran claims that "most serious limitations with user involvement arise from shortcomings of the representative structures put in place" [74]. The author regards the selection of representative users as crucial and suggests to aim at as genuine a representation of the user population as possible.

Over the last decades users have been viewed from several perspectives, depending on what software was predominantly used for and on the increasing prevalence of software systems. Kanstrup and Christiansen report on a gradual increase in user significance from "victims" in the 1970s, to "competent practitioners" in the 1980s, to "serious professionals" in the 1990s, and to a valuable "source of inspiration" in the 21st century [156]. Similar changes can be observed regarding the question whom to involve during software development. While it has been easy to identify system users in the 1970s, it is regarded as key challenge for both user involvement and requirements engineering today [25, 212, 263].

Theoretically, user representation ranges from *no users* to *all users* as illustrated in Figure 2.4. Practically, however, there are limitations. While back in the 1970s it was feasible to involve all prospective users of a future system in its development, this changed with the beginning of the following decade. Already in 1983, Gould and Lewis recommend to involve "typical users" during the design process in their seminal work on user-centered design [115]. Research accepted the practical limitations of user representation (e.g. [166]) and subsequent research – if any – rather concentrated on how to find the "right" set of users. However, the little research that was carried out in this regard belongs to product development, innovation, and marketing, and is typically concerned with customer integration in new product development (e.g. [96]) rather than with user involvement in system development [166].

Identifying the right user representation is a non-trivial task, which is particularly difficult for a large amount of heterogeneous users [166]. Grudin [121] reports on the following four challenges in identifying appropriate users. First, actual users of a product are often unknown until they buy it. Partially, this holds even true for new releases of existing software which often aim at expanding market share. A second challenge arises if product development aims at as broad a target group as possible. Third, the size of a development company complicates the identification of the right set of users. Due to the following subdivision of work, single software engineers seldomly see the big picture. Geographical distribution in global software engineering projects increases this challenge. Further, the gap between users and developers is typically larger in big companies. As fourth challenge the author mentions that products are often modified after being released but before arriving at the user. For instance value-added resellers may tailor products for a specific market. Again, this enlarges the gap between developers and the actual users of a system.

The lack of common guidelines towards a systematic selection of user representatives has brought researchers to put the cart before the horse. Muller et al. [198] compare 60 distinct user involvement methods (called "participatory practices" by the authors) regarding various attributes, among them also "group sizes" which refers to the number of users that are suitable for the corresponding method. The authors leave the choice of the right method – and thus of the right user representation – to the engineering team. Bekker and Long [24] compare five of these methods. The corresponding user representations range from a "representative set" to "all prospective users".

More than two decades after Gould and Lewis had introduced user-centered design, Kujala and Kauppinnen [166] propose a process for identifying and selecting relevant users, in particular for field studies and to gather user needs. The process is iterative and consists of the following five steps:

1. Brainstorm a preliminary list of users.
2. Describe the main user characteristics (including market size).
3. Describe main user groups and prioritize them.
4. Select typical and representative users from the groups.
5. Gather information from the users and redesign the user group descriptions according to the new information gathered.

The fourth step consists of selecting a subset of users. The authors explain that whenever a group of users has to be selected, it is important to employ a sampling strategy that produces a *representative* sample. If it is possible to list all users, a *random* sample can be drawn, given that the chance for subgroups to be included is known, and the sample size is large enough [166]. However, also random sampling does not always simply work by its own. For instance, when selecting users for field studies or usability tests, *stratified* sampling is appropriate, since the population needs to be divided into mutually exclusive groups corresponding to the main user characteristics [87, 166].

Bergvall-Kåreborn and Ståhlbröst [25] report on the following characterization of users originating from a joint perspective between product development and information system development.

- *Lead users* are vanguard users with needs that will become common in the future, who are motivated to contribute in order to get these needs met [281].
- *End users* do or will actually use the system.
- *User-representatives*, which might be single persons or groups, represent end users in the development team and act on their behalf.
- *Early adopters* (also called *first buyers*) use new technologies faster than typical users.

The authors remark that both lead users and early adopters might not be representative of the "general market segment for a system, but rather the opposite" [166]. The difference between lead users and early adopters lies in their motivation. The main motivation for early adopters is the early use of a system, while lead users want to contribute to its development.

Meanwhile, requirements engineering research had to solve the same problem. To elicit the requirements of a system, developers need to interact and collaborate with prospective users, which typically have a completely different background [49]. But because their input and feedback are needed to elicit requirements, the emerging gaps between users and developers have to be filled. Several techniques were proposed and evaluated in response to this situation and are in broad use today.

Using scenario-based requirements elicitation [49, 56] developers elicit requirements by observing and interviewing users. During the first step of this technique, developers identify *actors* – different types of users the system under development will support. Next, developers observe prospective users during their current tasks and create a set of *scenarios* for the functionality of the future system. Scenarios represent concrete examples of how the system under development will be used and provide means for developers to communicate with users. Cooper [69] proposes *personas* – user archetypes which should concretize the too abstract notion of 'user' and thereby improve product design practices. Personas consist of a precise description of hypothetical users, including their goals, and represent a user group during the complete design process [166]. However, persona development processes typically put more emphasis on detailing descriptions of typical users than on accuracy while identifying representative users [265]. Agile software development methodologies such as extreme programming [21] identify and represent users with *user roles* and *user stories.*

**Conclusion 3.** In theory, user representation might range from none to all users. In contrast, it is commonly believed that it is impossible to involve all users in software development. Consequently most user involvement methods comprise the selection of a group of "representative" users. To select this group is seen as the key challenge, also because the real users might not even be known beforehand. The effects of a selection failure are typically severe limitations.

### 2.2.3 Developer Role

The goal of user involvement is to improve the usefulness of a system by understanding the needs and expectations of users. As a consequence, developers need to gather information from users. The term developer role refers to what developers accomplish to elicit this information. Typically, it refers to a specific kind of interaction with the involved users.

In general, a developer's role in user involvement methods is to support the acquisition of knowledge regarding user's tasks, needs, and expectations. In this regard, the developer's goal is that of an empirical researcher, who wants to get insights about a specific phenomenon within a given population[2]. Depending on which method the researcher uses, there are generally two forms of contact with the studied subjects. First, the researcher might observe subjects without a direct interaction, or only analyze existing data. The second form includes a direct contact with the subjects. Consequently, we argue that the developer role in user involvement methods will be either *direct* or *indirect*, depending on whether the developer directly interacts with users or not. Bekker and Long [24] compare five design approaches and found three developer roles: the *expert*, who represents users' interests, the *facilitator*, who organizes meetings, plans agendas, and facilitates participants' contributions, and the *emancipator*, who tries to increase the possibilities for a weak group to have influence [31].

Only few researchers investigated the actual percentage of developers concerned with user involvement. In a survey about user-centered design practices, Vredenburg et al. [284] found that around 27% of the project staff were involved in user-centered design activities. Grudin claims that for "user involvement to be effective, most or all members of the development team must be committed to

[2]Note that *empirical measurement* is one of the principles of user-centered design [115]. Moreover, empirical research in software engineering for instance employs focus groups [164] which are also a popular user involvement method [284].

the approach" [121], mainly because iterative development requires longitudinal involvement and since usually some software needs to be touched in order to build and test prototypes.

The developers in question then need to establish a connection to the (previously selected) users. According to Grudin [121], several potential obstacles during this initial phase might lead to profound gaps between developers and users. Traditionally, product development companies would protect their developers from real users or customers. The reason lies in the fact that an organization usually cannot afford to loose precious development time of highly qualified personnel for customization requests of single users. As a consequence, developers get isolated from such requests and are faced with an often already prioritized list of generic improvements to benefit scores or large amounts of users. Representatives might be reluctant to let developers meet the real customers or users. They fear that developers, who are often seen as coming from a different "culture", might talk about development internals, thus creating dissatisfaction with current products, or even offend or alarm users or customers.

Furthermore, Grudin explains that developers often do not follow through even though they typically agree to the importance of user involvement in principle. The main reason lies in a number of gaps between users and developers, which can be of technical, social, and socio-technical nature. For instance, according to Grudin developers sometimes lack sympathy or empathy for inexperienced or nontechnical users. Users and developers often differ in terms of age, opinions, or academic background, and typically speak "different languages" regarding the product. In practice, developers often lack the experience to compensate for these differences, a soft skill which is hard to teach. According to Grudin, the "best of intentions can succumb to these factors, especially in the face of the slowness and imprecision that often accompany user involvement" [121].

**Conclusion 4.** Developers' roles in user involvement range from indirect observation to direct interaction. Research indicates that user involvement is most beneficial if it is regarded as common organizational culture by all developers. But as it turns out the relationship between users and developers is characterized by several gaps which eventually detach the organization from its users. Being a process of complex, *socio-technical* nature, user involvement thus bears several challenges for a effective information exchange between users and developers.

### 2.2.4 Procedure

A procedure specifies how knowledge about users' needs is practically acquired with a specific user involvement method. Bekker and Long [24] use the notion of "user involvement process" to indicate the actual steps to be followed. Similarly, Muller et al. [198] refer to a "process model". However, we argue that the notion of *process* has a longitudinal character, while *procedure* fits more our conceptual model of a user involvement *method*. Nevertheless, user involvement methods can be used repeatedly and in combination with other methods, thus aligning the single procedures to a user involvement process.

The procedure represents the main differentiator of user involvement methods. It answers questions such as how the involvement should be implemented, what participants do to communicate with one another, how they make decisions, and what they do with the materials used [24, 198].

As it turns out, it is not commonly agreed that a method oriented approach is suitable for user involvement [24], since single practitioners explicitly object to it (e.g. [34] and [37]). According to Muller et al. [198], there are two main reasons for this belief. First, the assumption that "a method is a straightforward, usually linear or sequential, series of well-understood steps that will lead to a predictable and relatively guaranteed outcome" [198]. Second, being able to name a method may incite engineers to believe that they have applied it in the right way, while the opposite may be the case.

We argue that methods should always be seen as a tool. The experienced practitioner knows how to apply a specific tool. In particular, she is aware that user involvement – as software engineering in general – is a complex, socio-technical, communication- and collaboration-intensive process where methods can never be simply applied with guaranteed success.

Grudin [121] explicitly reports on challenges related to the communication aspect of user involvement methods. First, the collected feedback does not always reach the developer. Developers may be shielded from external contacts by customer support groups who maintain products while working with customers on particular problems. Second, there might be no or not enough feedback.

> "The extent of feedback may vary with the pattern of marketing and product use. A company such as Apple, with a heavy proportion of discretionary purchases initiated by users rather than by management or information systems specialists, accrues benefits from having a particularly vocal user population. In general, though, a lack of user feedback may be the greatest hindrance to good product interface design and is among the least recognized defects of standard software development processes." [121]

Grudin explains that developers often lack experience with user feedback, which represents an obstacle to product improvement. Moreover, they often need to decide between alternatives based on metrics other than user feedback, typically time to develop and performance[3].

Similarly, Maalej et al. [181] elaborate on various communication gaps between users and developers for different user involvement procedures. The au-

[3]These decisions are called "design goals" in software engineering [49].

thors align the issues in “communication activities” according to the type of feedback (implicit or explicit) and the type of communication (push or pull). First, when feedback is explicitly gathered, communication is often inefficient and gathered information misinterpreted, since users often cannot articulate or do not remember their needs, simply do not know them, or forget important context information. Required efforts and resources to overcome these gaps are typically high, while human factors like subjectivity, interpretation, or social distinctions might absorb their effects. Second, feedback explicitly and proactively provided by users, frequently lacks important details and context information (cf. [298]). The underlying communication is often inefficient due to multiple iterations of requests and responses and requires users’ motivation to yield contributions of a sufficiently high quality. Third, if developers implicitly elicit user needs by analyzing legacy documents or by gathering usage data for product increments, they typically do not have any “back channels” to reach the users for clarifications and find themselves faced with a relatively large amount of noise and irrelevant information. Finally, lead-users might also actively deliver implicit information by thinking further than a user only interested in “consuming” features. According to Maalej et al. the particularly high chances in the software domain that such users might become competitors, might be a reason why the lead-user method is a rather uncommon user involvement method.

**Conclusion 5.** How user involvement is actually implemented is specified by procedures, which typically provide step-by-step instructions and explain how users and developers should communicate, on which basis decisions should be made, and which materials are necessary. Critics of a method-oriented approach underline the social component of user involvement, arguing that developers need experience and flexibility more than step-by-step descriptions, and that success is by no means guaranteed. Communication is seen as the most vulnerable part of any user involvement procedure, given various potential gaps between users and developers, in particular missing context information.

### 2.2.5 Types of User Information

Heiskari and Lehtola define user information as “anything that describes the users, their needs, problems they encounter, or the context they operate in” [136]. Interviewing six professional developers, the authors found seven different types of user information. Table 2.2 summarizes these types of user information together with their respective frequency. The study revealed that user feedback from already deployed products together with feedback from beta testers are the most common types of information. The generalizability of the quantitative results might be questionable. However, in particular for the purpose of this dissertation, it makes sense to further classify user information.

Table 2.2: Types of user information discovered by Heiskari and Lehtola (adopted from [136]).

| User information | Count |
|---|---|
| End user feedback | 6 |
| Beta feedback | 5 |
| Feature requests | 1 |
| Vision document | 1 |
| Usability problems | 1 |
| Conceptual requirements | 1 |
| Wholesaler profiles | 1 |

If directly mentioned at all, user involvement literature typically names "user needs", "user tasks", or "user problems" when referring to information gathered while working with users in a greenfield-engineering project (e.g. [116, 166]), and "user feedback" when referring to information gathered on existing systems or system prototypes (e.g. [24, 167]). We think the reason for these few described types is that user involvement literature typically falls into one of the four following categories.

1. *Problem statement.* Claims positive or negative effects of user involvement, argues about its definition or about missing empirical measures for its effectiveness (e.g. [146]).
2. *Study.* Analyzes effects of applied user involvement methods, such as its effectiveness, costs, or benefits (e.g. [167]).
3. *Meta-study.* Summarizes or compares different user involvement methods or studies (e.g.[24]).
4. *Solution proposal.* Describes a specific user involvement method (e.g. [116]).

Most publications belong to the first two categories, least to the third and fourth. On the contrary, specific instances of user information types are mostly introduced in solution proposals. For the purpose of this dissertation, we do not aim at a complete description of different user information types. We rather want to align the most common types in a preferably simple taxonomy, in order to provide a foundational glossary.

The main types of user information identified by Heiskari and Lehtola are different kinds of user feedback (end user feedback, beta feedback, and problem reports), feature requests, and design documents. According to Grudin [121], user feedback may be collected from "bug reports" and "change requests", notions which are also common in software maintenance and configuration management.

The software maintenance standard ISO/IEC 14767:2006(E) defines two information entities which may be provided by users. First, *modification requests* "identify proposed modifications to a software product that is being maintained" [144]. Modification requests, which are also referred to as *change requests*, are the primary input for software maintenance activities and may later be categorized into *corrections* or *enhancements*. Second, *problem reports* "identify and describe problems detected in a software product" [144]. A modification request may be created in response to a reported problem, but on the contrary also a problem report may be created while analyzing a modification request.

Similarly, configuration management [142] is concerned with management and controlling of change during software systems evolution [49]. To enable developers to deal with change, configuration management relies on a formal process to capture, analyze, and work on *change requests* (also called *requests for change*). Change requests are formal reports created by users or developers and denote the request to modify a work product [49].

Issue trackers are commonly used in software projects to report and follow problems with a software system or to request potential enhancements [12]. In the case of open source software the issue tracker is typically public, so that all users may create reports. Issue trackers store reported issues in tickets, which can be classified into different types, according to the reported issue. As a consequence, these types correspond to types of user information. We did not find any scientific publication on the different ticket types used in typical software projects. However, common issue trackers such as Jira[4] or Trac[5] are shipped with a predefined set of types, typically including *defect*, *enhancement*, *issue*, and *new feature*.

We argue that the main differentiator for user information types lies in the availability of the developed system. If this system is not yet developed, gathered user information is *prospective* regarding the system. Consequently, prospective user information typically describes current user tasks or user requirements. In contrast, user information is *retrospective*, if it is gathered from users who actually used the already existing system. Retrospective user information therefore usually takes the form of feedback. Table 2.3 distinguishes the different types of user information described in literature according to these categories.

In Figure 2.5, we illustrate user information types which are relevant for the purpose of this dissertation and how they are related to each other. *Requirements* typically represent prospective user information, while *feature requests* may represent prospective and retrospective user information. *Feedback on existing features* as well as *error reports* are necessarily retrospective. Note that the information type does neither determine *how* the information is obtained nor *in which form*. The focus of this dissertation is user involvement during software evolution, which is why we primarily focus on *retrospective* user information.

[4]http://www.atlassian.com/software/jira
[5]http://trac.edgewall.org/

Table 2.3: Different types of user information.

| Reference | Prospective | Retrospective |
|---|---|---|
| Heiskari and Lehtola [136] | conceptual requirement, feature request, vision document | beta feedback, end user feedback, feature request, usability problem |
| Grudin [121] | user need, user work, user task | bug report, change request, user experience |
| ISO/IEC standard for software maintenance [144] | — | change request, problem report |
| IEEE standard for configuration management [142] | — | change request |
| Ticket types in issue trackers [90, 149] | new feature | defect (bug), enhancement (improvement), issue (problem), new feature |
| Requirements engineering artifacts [49] | scenario, use case | feedback on prototypes |

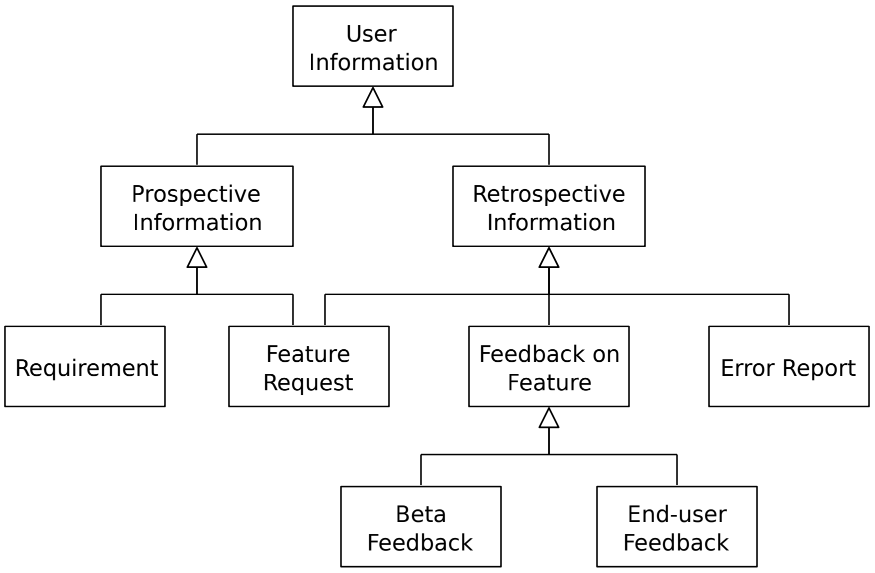


Figure 2.5: Taxonomy of user information types.

**Conclusion 6.** User information gathered through user involvement methods can be prospective or retrospective, depending on the availability of the system under development. Important user information types include requirements, feature requests, feedback on existing features, and error reports.

### 2.2.6 Time and Place

User information could be gathered throughout the whole software lifecycle, ranging from prospective information in the early stages of the project, to feedback on prototypes, and to continuously gathered retrospective information during software evolution. As already discussed, prospective user information is typically gathered in the beginning of a project, while retrospective user information usually during later phases. For instance, Bekker and Long [24] distinguish between *early* involvement, *late* involvement, and involvement *throughout* the lifecycle.

Similarly, the place where users are involved varies. Depending on the method utilized, but also on the phase of the software lifecycle, it may range from the *user environment* to the *development environment*. In practice, retrospective user information can be obtained with a specific external tool or service, but also directly while actually using the software. The latter case is commonly called "in situ" feedback and should be preferred according to several authors (e.g. [107, 182, 262]), since it avoids that users have to leave the application and interrupt their workflows in order to contribute.

Muller et al. [198] provide a comprehensive overview of different user involvement methods, and organize them according to both the time during the software lifecycle where they may be useful and the place where they are likely to be used. Table 2.4 summarizes user involvement methods suitable for a specific software lifecycle phase, while Table 2.5 depicts those applicable in multiple software lifecycle phases. Of the 60 methods[6] compared by Muller et al. [198], 35 are useful in early lifecycle phases (up to, including, object design), while 21 methods are suitable in later lifecycle phases (i.e. from evaluation onward). Of these methods, 14 are useful in both early and late lifecycle phases. The remaining 18 user involvement methods span multiple phases of the software lifecycle. As illustrated in Table 2.5, 10 of these multi-phase methods apply to early lifecycle phases, while the remaining 8 suit the later lifecycle phases. Similarly, Bekker and Long [24] compared 5 user involvement methods. The authors found that 3 methods are used in early phases while one of them is also used in later stages. The remaining 2 methods are used throughout the development process.

Kujala [165] claims that user involvement is most beneficial in the early stages of system development, as later the costs to make changes increase. We do not

[6]Note that Muller et al. originally list 61 participatory practices. However, *organization game* and *layout, organization, and specification games* refer to the same method.

Table 2.4: User involvement methods organized by time and place (adopted from [198]).

| | Prospective | | | | Retrospective | | |
|---|---|---|---|---|---|---|---|
| | Problem identification & clarification | Requirements engineering | System design | Object design | Evaluation | End user customization | Re-design |
| User environment | Forum Theatre [159]<br>Ethnography [33]<br>Mock-Ups [91]<br>Search Conference [224]<br>Starting Conference [224] | Artifact Walkthrough [290]<br>Blueprint Mapping [161]<br>CARD [200]<br>Ethnography [33]<br>Forum Theatre [158]<br>Mock-Ups [91]<br>PICTIVE [197]<br>PictureCARD [274] | Artifact Walkthrough [290]<br>Lunchbox [254] | Artifact Walkthrough [290] | Forum Theatre [159]<br>Interface Theatre [201] | Buttons [186] | |
| Intermediate environment | Future Workshop [158]<br>Graphical Facilitation [71]<br>Layout, Organization, & Specification Games [92]<br>Scenarios [56]<br>Storytelling Workshop [117]<br>Translators [287] | CISP [187]<br>Collab. Design Workshop [36]<br>Coop. Reqmts. Capture [184]<br>CUTA [168]<br>Future Workshop [158]<br>Graphical Facilitation [71]<br>Layout, Organization, & Specification Games [92]<br>Participatory Ergonomics [211]<br>Scenarios [56]<br>Translators [287]<br>Work Mapping [276] | ACE [85]<br>CARD [200]<br>Graphical Facilitation [71]<br>Metaphors Game [201]<br>Mock-Ups [91]<br>Scenarios [56]<br>Translators [287] | CISP [187]<br>Collaborative Design Workshops [36]<br>Critics [188]<br>Graphical Facilitation [71]<br>Icon Design Game [201]<br>Scenarios [56] | CISP [187]<br>Collab. Design Workshop [36]<br>Cooperative Evaluation [292]<br>Mock-Ups [91]<br>Participatory Ergonomics [211]<br>Participatory Heuristic Evaluation [199]<br>Pluralistic Walkthrough [26]<br>Scenarios [56]<br>Storyboard Prototyping [11]<br>Translators [287]<br>Work Mapping [276] | Critics [188] | Critics [188]<br>Priority Workshops [45] |
| Development environment | Workshop for O-O GUI Designing [194] | KOMPASS [119]<br>Prototyping [51]<br>TOD [194]<br>Workshop for O-O GUI Designing [194] | HOOTD [42]<br>KOMPASS [119]<br>PrOTA [198]X<br>Prototyping [51]<br>TOD [194]<br>Workshop for O-O GUI Designing [194] | BrainDraw [84]<br>HOOTD [42]X<br>PICTIVE [197]<br>PrOTA [198]<br>Prototyping [51]<br>Video Prototyping [294] | CARD [200]<br>PICTIVE [197]<br>Prototyping [51]<br>TOD [194]<br>Workshop for O-O GUI Designing [194] | | |

Table 2.5: User involvement methods spanning multiple lifecycle phases (according to [198]).

| | Prospective | | | | Retrospective | | |
|---|---|---|---|---|---|---|---|
| | Problem identification & clarification | Requirements engineering | System design | Object design | Evaluation | End user customization | Re-design |
| ACOST [76] | ✓ | ✓ | | | ✓ | | |
| CESD [118] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Codevelopment [10] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Conceptual Toolkit in CSCW Design [37] | | ✓ | ✓ | | | | |
| Contextual Design [289] | | ✓ | ✓ | | | | |
| Contextual Inquiry [138] | ✓ | ✓ | ✓ | | | | |
| Diaries [44] | | | ✓ | ✓ | | | |
| ETHICS [203] | | ✓ | ✓ | | ✓ | | |
| FIRE [46] | ✓ | ✓ | ✓ | | | | ✓ |
| Florence [30] | ✓ | ✓ | ✓ | | | | |
| Group Elicitation Method [43] | ✓ | ✓ | | | | | |
| Hiser Design Method [35] | | ✓ | ✓ | | | | |
| JAD [291] | | ✓ | ✓ | ✓ | | | |
| ORDIT [125] | ✓ | ✓ | | | | | |
| SSADM [177] | | ✓ | ✓ | | | | |
| SSM [63] | ✓ | ✓ | | | ✓ | | |
| STEPS [101] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| UTOPIA [38] | ✓ | ✓ | ✓ | | ✓ | | |

fully agree with this claim, as it seems to assume a simple sequential development process such as the waterfall model [249]. Although it is true that change typically produces higher costs the later it occurs [49], this should not be an excuse to develop an unusable or useless system. More recent development processes, such as the spiral model [41] or agile processes like Scrum [261] embrace change and provide facilities to iteratively obtain feedback and react to it. Moreover, when developing mobile applications, not all possible contexts of use can be checked or simulated before shipping the application. Consequently, user involvement will become even more important in the later stages of the software lifecycle for such applications.

**Conclusion 7.** User involvement methods have been proposed for early and late phases of software development processes. Depending on the state of the system under development, developers gather either prospective or retrospective user information. Research mainly investigates the benefits and challenges of early user involvement, since it is commonly accepted that changes become more expensive over time. While we agree with this fact, we claim that the yet undiscovered field of user involvement during software evolution bears huge potential. In-situ feedback should be preferred.

## 2.3 User Involvement Effects

Theoretically, involving users in the development of software should lead to increased probability of software success and higher user satisfaction [165]. Practically, various different, partly contradictory, results have been reported (e.g. [135] and [167], see also [146]). Experiences with user involvement have been described in industrial studies, which illustrate effects of applied user involvement and meta-studies, which summarize and compare existing results from user involvement research and practice. We summarize these experiences by pinpointing the reported benefits and challenges.

### 2.3.1 Benefits

Kujala [165] conducted a meta-study to understand early user involvement and its effects in practice. The author described various benefits, particularly on system success and user satisfaction. Specifically, developers experienced to get "more accurate user requirements", avoiding "costly" features which were not wanted or needed by users. User involvement led to higher probabilities of system acceptance and increased user satisfaction. In a survey with over 200 product managers, already Baroudi et al. [19] had found that user satisfaction in the consequence also led to greater system usage. Later, Kujala et al. [167] empirically investigated the role of user involvement in typical development projects with respect to the definition of user requirements. Surveying 18 practitioners from 13 Finnish companies, the authors could confirm that user involvement led to better requirements quality. Further, projects where developers had based requirements on user information had a significantly higher probability of success.

Vredenburg et al. [284] surveyed over a hundred user-centered design professionals to investigate how user-centered design methods were used in practice across the industry. A majority of around 80% of the participants agreed that user-centered design methods have improved usefulness and usability of their products. The study also indicated that not only benefits are taken into consideration by developers. Rather the relation between costs and benefits determine if methods are applied and if they are considered helpful. Consequently, the most commonly used methods should be effective and cheap. The five most important user-centered design methods found by Vredenburg et al. are, in descending order, field studies, user requirements analysis, iterative design, usability evaluation, and task analysis. On the other hand, the five most used methods are iterative design, usability evaluation, task analysis, informal expert review, and field studies. The fact that these lists are not equal reveals there is still potential for more cost-effective methods. Furthermore, the authors asked survey participants to name measures of effectiveness for user-centered design methods. Among the most frequent results were external satisfaction, enhanced ease of use, impact on sales, and reduced helpdesk calls. Accordingly, successfully applying user-centered design should lead to better usability, increased sales numbers, and less maintenance efforts.

**Conclusion 8.** Several studies have shown that user involvement can have positive effects on system success and user satisfaction. Specifically, increased user requirements quality as well as improved usability and usefulness have been reported, leading to increased sales numbers and less maintenance efforts. However, in practice both cost and benefit of user involvement determine its applicability. Research has shown that there is still potential for more cost-effective user involvement methods.

## 2.3.2 Challenges

Rosenbaum et al. [245] surveyed over 130 HCI professionals to understand how organizational approaches and usability methodologies contribute to the strategic impact of usability research within companies. Major obstacles found by the authors include resource constraints, resistance to user-centered design or usability, as well as lack of knowledge about usability. Vredenburg et al. [284] conducted a similar study and revealed that measurements of effectiveness and common evaluation criteria across the industry are missing. This finding intensifies the obstacles identified by Rosenbaum et al., since it objects the acceptance of user-centered design in practice. Consequently, the authors suggested that cost-benefit tradeoffs play an important role in the practical adoption of user-centered design methods. For instance, while field studies were ranked high on practical importance, companies seldomly used them due to their high costs.

Grudin [121] identified communication gaps between users and developers in the 1990s, which were confirmed even 20 years later in a study by Heiskari and Lehtola [136]. In particular, the authors revealed that there was little interaction between users and developers. In addition, if users provided information, it did often not reach developers. As a result, too little user information is present in software companies. Heiskari and Lehtola additionally found that this information is mostly scattered across different tools, unorganized, difficult to access, and not integrated into existing development processes. As further consequence, developers cannot assess what users actually want, or more specifically, their average opinion. Two factors further complicate such an assessment. First, users often have contradictory expectations, leading to more different opinions to estimate. Second, typically only a subset of users is actually involved, making it difficult to obtain a significant estimation.

Kujala et al. [167] pointed out an additional problem when gathering user requirements. Even with the best of intentions, users might not be able to express their problems and needs, since part of their knowledge has become tacit through automation. Grudin [121] further pointed out that in practice there are always other forces which shape software when users are not involved, or when accessing the relevant knowledge is too difficult for developers. In many cases, these are even more directly connected with measurable and influential properties of a software project, specifically budget and time. As a consequence, users' voices often remained ignored. Gould and Lewis [116] had already identified that user-centered principles were often not followed in practice. One major reason according to the authors was the misestimation of its value, for instance due to companies' belief that users do not know what they want.

Heinbokel et al. [135] investigated effects of user participation and user orientation, and found that both were negatively affecting process and product quality. From their explanations, we conclude that users in the studied scenarios were given too much decisive power. For instance, developers needed to change the design completely before delivering the software due to a late intervention by a user. However, this study pinpoints that a major challenge lies in finding the right mix of degree, time, and other user involvement dimensions. Only with such a mix user involvement will have positive effects.

**Conclusion 9.** Studies have revealed three major challenges for user involvement in practice. First, resource constraints often do not allow companies to employ effective user involvement methods. Second, measures of user involvement effectiveness are missing, what prevents practitioners from assessing cost-benefit tradeoffs. Third, there are a number of communication gaps between users and developers. These can lead to little user information, which might be scattered and difficult to access for developers. As a consequence, other forces than users' needs and expectations shape software in practice.

## 2.4 Summary

User involvement in software engineering has become an established research field [165], which is studied particularly at its intersection with human-computer-interaction [24]. Its foundations go back to the early 1980s, in particular to Gould and Lewis' key principles of usability [115] which together with Norman and Draper's work [209] became the pillars of user-centered design, and Ives and Olson's work on the effect of user involvement on system success [146], one of the first critical studies of user involvement in practice. User involvement aims at maximizing the usefulness of a system by understanding users' needs and expectations. User involvement methods, which provide for a systematic information exchange between users and developers, include multiple, social and technical dimensions. During our literature review we found the following important trends.

- User involvement has changed significantly over the last three decades, mainly due to the radical progress of hardware and software. At the same time software users changed from programmers or trained technical staff to practically any person [120], causing a fundamental shift in developers' attitude towards them [156]. With the advent of application distribution platforms and mobile devices, neither users nor the complete use context of software is known before actually delivering it. The consequence of this evolution is an increasing distance between developers and users [121], while more focus on users would actually be necessary to satisfy their increasing demands [182].

- Various methods to involve users in software development have been proposed [198], which provide different levels of decisive power to the involved users [74]. However, in research and practice there is no common agreement on the "optimal" degree of users' influence on the developed system [146], and probably never will [121]. Consequently, researchers have published comparative studies and overviews to support practitioners in their choices [24, 198]. Practical forces influencing design and development are typically time and budget rather than user satisfaction [121].

- Most user involvement methods comprise the selection of a group of "representative" users, since it is common belief that involving all users is impractical or impossible [166]. Selecting the right users is regarded as key challenge for user involvement and requirements engineering [25, 212, 263], with a wrong selection typically leading to serious limitations [74]. Due to the restrictions on user representation and the complex, socio-technical nature of user involvement in software development, information exchange between users and developers is characterized by several gaps (e.g. [105, 120, 181]).

- User involvement methods have been proposed for early and late phases of software development processes, fewer even span multiple lifecycle phases [198]. Depending on the state of the system under development, developers gather either prospective (early) or retrospective (later) user information [136]. Research mainly investigates the benefits and challenges of early user involvement [165] often with a focus on participatory practices, since it is commonly accepted that changes become more expensive over time [49]. While we agree with this fact, we claim that the yet undiscovered field of user involvement during software evolution bears huge potential. Specifically, there seems to be little research on how retrospective user information, i.e. user feedback, influences development and which challenges and benefits it provides.

- Research has shown that user involvement can have positive effects on system success and user satisfaction [165]. Specifically, increased user requirements quality as well as improved usability and system usefulness have been reported, leading to increased sales numbers and less maintenance efforts [284]. However, in practice both cost and benefit of user involvement determine its applicability, since resource constraints often do not allow companies to employ effective user involvement methods [284]. But measures of user involvement effectiveness are missing, preventing practitioners from assessing cost-benefit tradeoffs [284]. The various communication gaps between users and developers can further decrease user involvement effectiveness [121], leading to little user information, which might be scattered and difficult to access for developers [136]. As a consequence, other forces than users' needs and expectations shape software in practice [121]. This shows that there is still potential for more cost-effective user involvement methods.

Most existing research about user involvement is concerned with gathering user information in the early phases of the software development lifecycle. Numerous studies have been carried out to explore the effectiveness of corresponding methods. User-developer communication is essential during the early phases of software projects, but phenomena such as IKIWISI [39] – "I'll know it when I see it" – indicate that it should not end there. While Boehm has made remarkable groundwork introducing iterative design [41] and a project management theory [40] to deal with these phenomena throughout the whole software lifecycle, there is little research exploring how user involvement is applied in software *evolution* practice, and which benefits and challenges it provides.

As mobile and ubiquitous computing is becoming mainstream, user requirements and contexts of use are less and less known upfront. At the same time, users are becoming more demanding and exigent, while the internet and Web 2.0 in particular allows them to be more extroverted regarding their needs [182]. With application distribution platforms, software change frequencies increase, and software becomes quickly available to practically any person. We claim that it is therefore necessary to investigate how user involvement theory applies to such scenarios, which problems developers experience and how they could be supported. In the next chapter, we explore the current practice of user involvement during the evolution of software

# Chapter 3

# Empirical Analysis of User Involvement in Software Evolution Practice

«*Tables of primes constructed in 1776 by Antonio Felkel were considered so useless that they ended up being used for cartridges in Austria's war with Turkey.*»

— Marcus du Sautoy, *The Music of the Primes*

In the previous chapter, we have analyzed the foundations of user involvement as reported in literature. User involvement is a complex socio-technical process. While we have found many methods to involve users, there seems to be no consensus about the effectiveness of user involvement. Although there are studies about user involvement in user-centered design projects, little is known about how and why users are involved during software evolution, and which are the benefits and challenges.

This chapter reports on an empirical case study which we conducted over three months with the goal to analyze the current practice of user involvement during the evolution of software especially with large user audiences. In particular we wanted to understand how and why users are involved in practice. Our goal was to find out what happens with user feedback in the development environment, and what developers' reasons and needs are. First and foremost, we aimed at identifying problems and challenges in the current practice and workflows, but we also wanted to glimpse at the practitioners' requirements on tools which would solve these issues.

Section 3.1 explains the study setting in terms of research questions and methodology, and introduces the research data. The following three sections summarize our research findings on the user involvement setting (Section 3.2), developers' workflows to analyze user feedback (Section 3.3), and developers' requirements for and expectations of tool support (Section 3.4). Section 3.5 discusses implications of our findings for researchers, practitioners, and tool designers and summarizes the limitations of our study, while Section 3.6 explores related work. Section 3.7 summarizes our study.

## 3.1 Study Setting

We first summarize the questions that drive our case study. Then, we describe the overall methodology we use to study user involvement in software evolution practice. Finally, we give an overview of the subjects we study.

### 3.1.1 Research Questions

The goal of this case study was to understand how professional software developers involve users during software evolution, which problems they encounter, and which benefits could be gained by resolving the encountered problems. We were particularly interested in three aspects: their user involvement *setting*, their *workflows* to analyze and work with user feedback, and their *requirements* for a tool-supported consolidation and assessment of user feedback.

**RQ 2.(a) User involvement setting** describes *how*, *where*, and *when* users may provide feedback. In particular, we aim at answering the following questions:

- *Infrastructure*: Over which channels do developers gather user feedback?
- *Frequency*: How often are users allowed to provide feedback?
- *User-developer communication*: Are users systematically involved?

**RQ 2.(b) User involvement workflow** details how and why developers *work* with user feedback. We aim at identifying and understanding developers' problems, answering the following questions:

- *Motivation*: Why is user feedback important?
- *Analysis*: How do developers analyze user feedback?
- *Problems*: Which problems do developers encounter, and which role do quantity, quality, structure, and content of user feedback play?

**RQ 2.(c) User involvement requirements** summarize developers' *needs* and *expectations* regarding tool support for user involvement. We focus particularly on user feedback consolidation and assessment, answering the following questions:

- *Tool support*: Do developers think that tool support for feedback consolidation could improve user involvement practices?

- *Consolidation*: How should user feedback be consolidated?
- *Assessment*: Would developers embrace user and usage benchmarks supplied by user feedback?

We formulated a set of 20 specific interview questions for these aspects together with a set of answers to facilitate the interview process. Additionally, we collected 7 meta-questions on the specific software project and about the subject's background, in order to enable a descriptive classification. The complete catalogue of questions can be found in Appendix A.

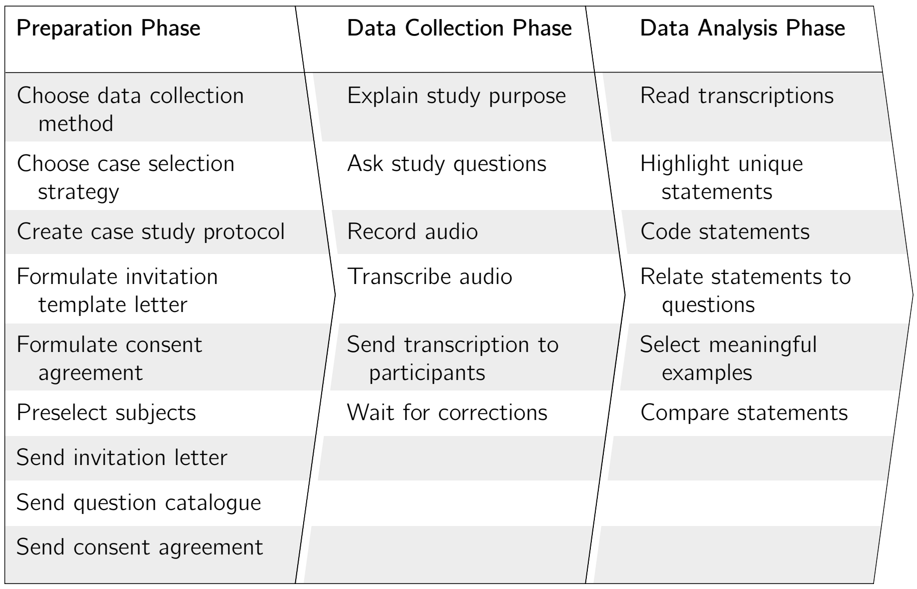
### 3.1.2 Research Method

Case studies can provide a detailed view on real-life situations and thus may reveal the reasons and mechanisms why and how specific phenomena occur [87, 102]. Our goal was to explore user involvement in software evolution practice to understand developers' current workflows, their problems, and their motivation to involve users. Additionally, we sought to describe and explain the situation whenever possible. Consequently, our case study mainly served an *exploratory* purpose, but sometimes exposes *descriptive* or *explanatory* character [250].

Our *study proposition* was threefold. First, we thought that developers appreciate user feedback as important source of information about the acceptance of their software. Second, we hypothesized that current user involvement practice is unsystematic and bears several technical challenges and practical limitations in particular due to the *quantity*, *quality*, *structure*, and *content* of user feedback [217]. Third, we expected that developers embrace tool support which allows them to deal with large amounts of user feedback and to assess the acceptance and status of their software according to its users.

To collect qualitative data which helps to explore the practical situation and to test our proposition, we conducted semi-structured and open-ended interviews [242] with professional developers working at different software companies. Semi-structured interviews allow for improvisation and thus facilitate an exploration of the studied cases [250]. Rather than randomly choosing study cases, we selected few software companies which develop and maintain *interactive* software for a large number of users, and one company with a smaller user audience in order to study potential effects of the amount of users. The selected companies allow their users to provide feedback and are interested in understanding it. We employed this *purposive* sampling strategy since we expected the resulting cases to be most relevant to our proposition [87].

For our case study, we followed the design principles and procedures described by Runeson and Höst [250]. As shown in Figure 3.1, our methodology consisted of three phases: a preparation phase, a data collection phase, and a data analysis phase.



| Preparation Phase | Data Collection Phase | Data Analysis Phase |
|---|---|---|
| Choose data collection method | Explain study purpose | Read transcriptions |
| Choose case selection strategy | Ask study questions | Highlight unique statements |
| Create case study protocol | Record audio | Code statements |
| Formulate invitation template letter | Transcribe audio | Relate statements to questions |
| Formulate consent agreement | Send transcription to participants | Select meaningful examples |
| Preselect subjects | Wait for corrections | Compare statements |
| Send invitation letter | | |
| Send question catalogue | | |
| Send consent agreement | | |

Figure 3.1: User involvement in software evolution – case study methodology.

#### 3.1.2.1 Preparation Phase

After choosing the data collection method and case selection strategy, we created a case study protocol in order to capture design decisions and document the data collection process. Next, we formulated a template letter to invite participants as well as a consent agreement to inform subjects about the study purpose and their anonymity. The consent agreement also documents the explicit agreement of the subjects to participate in the study. We sent the invitation letter per email to six selected companies which were possibilities and waited for the invitees to respond. Five developers agreed to the interviews. Around one week before an interview, we provided the specific subject with the question catalogue. We explicitly called their attention to the semi-structured nature of the interview, and underlined that the questions should be regarded as support.

#### 3.1.2.2 Data Collection Phase

Before starting each interview, we briefly explained the study purpose to the participant and recalled the exploratory nature of the study. As suggested by Runeson and Höst [250], we recorded each interview in an audio format after having asked for the interviewee's consent. The interviews took between 29 and 55 minutes, during which we asked the pre-defined questions as well as questions as a reaction to each subject's particular answers. After the interviews, we transcribed the recorded audio with the help of the notes we had taken during the interviews. We then sent each transcription to the corresponding interviewee and asked for corrections, which we did not receive.

Table 3.1: Overview of interview partners and studied projects.

| ID | comp. | w. exp. | project role | platform | user audience | # active users | rel. cyc. |
|---|---|---|---|---|---|---|---|
| S1 | C1 | 6–10 | developer | both | sp. consumers | > 10,000 | 3 wks |
| S2 | C2 | 3–5 | developer | desktop | all | 150,000–200,000 | 4 wks |
| S3 | C3 | 3–5 | developer | mobile | all consumers | > 10,000 | 2–6 mos |
| S4 | C4 | 3–5 | architect | mobile | all consumers | 100,000–200,000 | 2–6 mos |
| S5 | C5 | 6–10 | prod. mgr. | desktop | sp. professionals | 100–500 | 4–8 wks |

#### 3.1.2.3 Data Analysis Phase

We started analyzing the interviews by reading the transcriptions and highlighting remarkable and particularly unique statements. Due to the semi-structured nature of the interviews, the transcriptions did not always follow the order of the pre-defined question catalogue. Moreover, subjects often complemented prior statements as the interview proceeded, so that especially helpful details were frequently given during later questions. We therefore decided to code the transcriptions according to two schemata. First, the pre-defined question each statement was answering, and second, the research question for which we regarded it as relevant. The second schema was especially helpful to identify meaningful examples for each research question. We then answered the research questions by comparing the different statements.

### 3.1.3 Research Data

Five software companies agreed to take part in our case study. We interviewed one participant per company. Each participant reported on one specific, closed source, commercially successful project developed in her company. Table 3.1 gives an overview of the 5 participants and the studied projects. The column *w. exp.* denotes the work experience of a participant in years, the column *project role* her main role within the project.

Two subjects (S2 and S5) reported on *desktop* applications and two (S3 and S4) on *mobile* applications, while S1 reported on an application developed for *both* platforms.

In three cases (S1, S3, and S4) the users of the studied software are *consumers*, a *special consumer group* in the case of S1, and *all consumers* in the case of S3 and S4. From the remaining two subjects, S2 reported on an application with a *general* user audience, while S5 develops software for a *special group of professionals*.

Four of the five studied software projects (S1–S4) had a large number of users (more than $10,000$), whereas the software referred to by S5 accounted for a smaller user audience (less than 500).

Table 3.2: User involvement infrastructure: user feedback channels and frequency of utilization – ■ most often, ▣ often, ▨ sometimes, □ rarely

| | Channel | Distribution |
|---|---|---|
| Error Reports | Integrated | ■■■▣▣ |
| | Email | ■■■□□ |
| | AppStore | ■■ |
| | Phone | ▣□□□ |
| | Blog | ▣ |
| Feature Requests | Email | ■■▣▨□ |
| | AppStore | ■■▣ |
| | Integrated | ■▣▣ |
| | Blog | ▣ |
| | Forum | ▣ |
| | Twitter | ▣ |
| | Phone | ▨□□ |
| | Facebook | ▨ |

| | Channel | Distribution |
|---|---|---|
| Feedback | Email | ■■▣▨□ |
| | AppStore | ■■▣▨ |
| | Integrated | ■▣ |
| | Phone | ▣□□ |
| | Blog | ▣ |
| | Forum | ▣ |
| Ratings | AppStore | ■■■ |
| | Rating site | ■ |
| | Email | ▣ |

## 3.2 User Involvement Setting

Our interviews show that gathering user feedback plays an important role for the studied subjects. All subjects report that their users may contribute using *error reports*, *feature requests*, and *feedback on existing features* which includes *improvement* and *enhancement requests*. All subjects but S5 also regard users' *ratings* (e.g. in application distribution platforms such as the Apple AppStore [13]) as important user feedback. While being aware of their importance in other scenarios, S5 stated that ratings are irrelevant for the product she reported on. Interestingly, S5 reported on the only software being developed for a *smaller* group of *professionals*, what suggests that public ratings are less important in such contexts. In contrast, S1, who develops a product for a large consumer audience, regards ratings as the "most critical" user feedback which even can "create tension" and "harm your product". In particular if errors occur, users tend to quickly give bad ratings, which in turn immediately affect sales numbers. S1 illustrates the possible effects: "If you mess things up, they [users] will kill your product".

### 3.2.1 Infrastructure

Table 3.2 shows the user involvement infrastructure as reported by the subjects in our case study. For four types of retrospective user information we asked subjects to name the channels over which this user information reaches them, together with its relative frequency. In total we collected 9 different channels, which shows that user feedback is currently widely scattered rather than being a single source of information for developers. The top three channels for *error reports*, *feature requests*, and *feedback on existing features* are common except for their order. *Emails*, the *AppStore*, and mechanisms *integrated* in the software (for instance a feedback library) account for the most user feedback at subjects' companies. Ratings are mostly done in the AppStore, on particular rating sites, and via email, while integrated mechanisms are not used. However, subject S4 explained that their software prompts the user to rate the product after a specific time period, which seems to be common practice for mobile applications. None of the companies provides all users with access to a public issue tracker. Nevertheless, S3 explained that at her company a public issue tracker is utilized only in the case of one single software, whose users are professional salesmen. Apart from this exception, all subjects agreed that issue trackers are best for internal use only.

**Hypothesis 3.1.** *User feedback is scattered across multiple channels, with email, application distribution platforms, and integrated feedback mechanisms being frequently utilized.*

Interestingly, users seem to select the appropriate channel *intentionally.* S1 describes: "The more critical their feedback is, the more public is the channel they choose." She reported on a case where the company had to concentrate on the mobile version of the software, and decided to delay the development of some of the desktop version's features. "At the beginning we received single mails, but as more and more desktop users felt left over, they started a public campaign on Facebook." S2 confirmed this user behavior: "We had a user forum but discontinued it, because the users allied to request features we did not want to implement." In particular errors are immediately published. According to S1, users tend to give low ratings to apply pressure: "It creates a lot of tension if your users write 'you'll keep getting one star until you fixed X'."

**Hypothesis 3.2.** *Users intentionally select the feedback channel to apply pressure by allying against the software company. The more critical their feedback is, the more public is the channel they choose.*

### 3.2.2 Frequency

In all studied companies, users may provide feedback continuously. In practice this means that the feedback channels remain continuously open. Subjects S1, S2, and S4 consequently report on a continuous stream of user feedback, with dozens of messages per day. S2 counted around 300 messages per month, if no new version is released. She further described that typically only very few users provide feedback more than once, while the majority reports rarely or just once.

Table 3.3: User-developer communication modes across studied companies.

| Feedback type | C1 | C2 | C3 | C4 | C5 |
|---|---|---|---|---|---|
| Error reports | **pull** | **pull** | push | push | **pull** |
| Feature requests | **pull** | push | push | push | push |
| Feedback | push | push | push | push | **pull** |
| Ratings | push | push | push | **pull** | push |

On the other hand, C3 and C5 receive less feedback, for two different reasons. C3 develop software on behalf of other companies. Therefore, they typically get presented with feedback digests, pre-selected user feedback, or feedback reported by their customer. S5 on the other hand explained that their users, who all use the software professionally, first collect several ideas and feedback and then send one single message which contains them all. Consequently, C5 receives feedback less frequently but in high concentrations.

In general, users seem to give feedback just as their concern happens. Subject S1 explained that therefore user feedback and product backlogs might contradict each other: "The question is always when and to what extent do you consider the feedback. But if they really shout, you need to react quickly."

**Hypothesis 3.3.** *Users frequently provide feedback, but user feedback does not always reach developers.*

### 3.2.3 User-Developer Communication

None of the studied companies employed focus groups or a similar user involvement method to gather user feedback after the product launch, for instance to assess a planned new feature. Only C3 had conducted focus groups in the beginning of single business-to-business projects, but with their customers instead of end users. S3 explained that focus groups were regarded helpful to get to know a new application domain. Moreover, S3 claimed it was common practice that end users are not involved until the launch of a product, while customers typically take the role of the end user during the initial development phase. However, customer involvement seems to serve a purpose other than eliciting user requirements, as S3 justified this practice stating that "customers know what they want, but not how it should look like".

User-developer communication is established differently across all studied organizations, as illustrated in Table 3.3. *Error reports* are automatically triggered in the case of C1, C2, and C5, where users are asked to provide their error reports after a crash has occurred. C1 explicitly trigger *feature requests*, as S1 describes: "Sometimes we ask our users on Facebook what they desire, and usually get constructive suggestions." *Feedback on current features* is explicitly requested only in the case of C5, who role out beta versions to selected users. Finally, C4 explicitly trigger product *ratings* after fixed time periods by redirecting the user on a specific AppStore page.

An interesting observation was made by S5, who distinguished four different communication types. First, conversations with users at fairs are triggered by the users and may eventually lead to new product ideas. Second, beta versions made available to selected users trigger frequent, personal, bi-directional discussions. Third, users proactively ask questions on features in pre-sales software versions they have used for a *short* time. Finally, the typical "support communication" regards users' experience with a product which they have bought and used for a *longer* time.

The most systematically supported feedback type seem to be error reports, as all studied products are capable of including automatically generated stack traces and usage data as an error occurs. The reason may lie in the very concrete nature of errors. It is evident when they happen, and quite straightforward to automatically collect related information with established development frameworks or libraries. All other user feedback is not "machine-readable" and in particular lacks a tangible trigger. As a consequence, there is no common practice neither on providing nor on gathering such feedback. For instance, S5 explained that their users include multiple suggestions in one message and often even mix different feedback types. On the other hand, companies do not systematically "educate" their users to a common, helpful way of giving feedback.

**Hypothesis 3.4.** *Users are not systematically involved during software evolution. Apart from error reports there is no commonly agreed practice neither how to* provide *nor how to* gather *user feedback during software evolution.*

## 3.3 User Involvement Workflow

When investigating their user involvement workflow, we asked our subjects why and how they work with user feedback and which problems they encountered.

### 3.3.1 Motivation

The main motivation to appreciate user feedback seems to be its origin: the user. Our interviews revealed that companies are interested in their feedback particularly because of two reasons. First, companies are interested to satisfy user needs, since eventually users buy their products. Users' goodwill can quickly turn into anger, and even harm the company, if they get frustrated with the software. Consequently, software companies continuously seek to *assess* the *acceptance* of their products. Second, developers need *real-world data* from users' environments, be it statistics about which feature is used or which errors occur most and in which context. Such data is especially helpful to complement software tests and to align development efforts with feature importance.

**Hypothesis 3.5.** *User feedback supports the continuous assessment of product acceptance and serves as real-world usage data.*

All subjects agreed that user feedback is helpful to reach three main goals: to improve software quality, to identify missing features, and to advertise and market a product.

**Improve Quality** Users typically help to improve software quality by reporting latent errors. S4 explains that problem reports are the feedback they are most interested in: "Often we get problems which we couldn't think of before, since users have very heterogeneous configurations on their machines." S5 further illustrates the value of error reports: "We need crash reports and stack traces. Without them we would not know which crashes happen out there." In single cases, products were even released with few presumably insignificant known bugs. But they turned out significant instead, what eventually affected ratings and sales numbers.

**Hypothesis 3.6.** *User feedback helps to improve software quality.*

**Identify Missing Features** We found that users of all studied companies frequently request additional features. On the one hand, companies appreciate this feedback as it helps to perfect a product. S1 illustrated that missing features affect product acceptance: "From their comments we could see that they will never accept our product without this feature." On the other hand, companies need to be ahead of their users and cannot create products by simply reacting to their feedback. Instead, products are developed following internal roadmaps, while external feature requests are only regarded as additional support and signposts. S4 named one important reason for this: "We filter very wisely, because we do not know how many people this will really help." This information need was also confirmed by the other subjects. As a consequence, small, incremental, perfective feature requests are rather heard than revolutionary new ideas. S5 explained that professional software users are aware of this fact: "Our users thoroughly make up their mind what could improve their workflows and increase their profit. Then they argue that it would help other users as well."

**Hypothesis 3.7.** *User feedback helps to identify missing features, but developers need to assess how many users will benefit from a specific new feature.*

**Advertise and Market Product** All subjects agreed that in particular two types of user feedback benefit their marketing. First and foremost, having many positive ratings in application distribution platforms and particular rating sites pushes applications into top lists, which in turn leads to more downloads and higher sales numbers. S4 illustrated that this was the reason why they explicitly

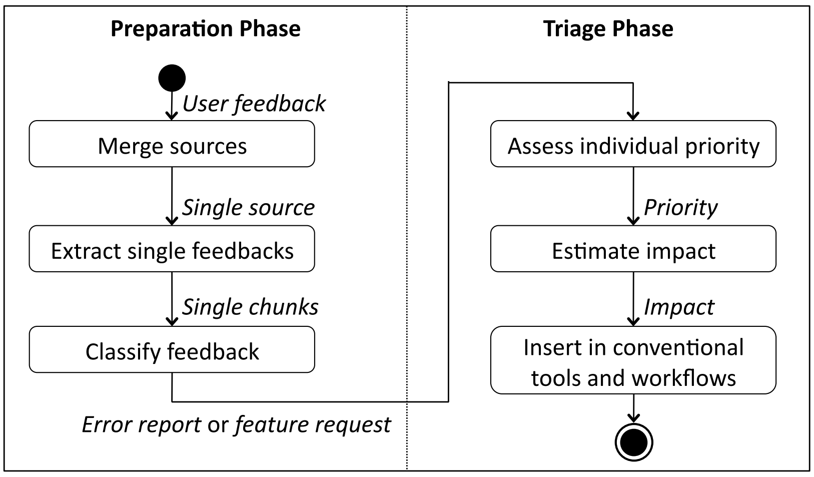


Figure 3.2: User feedback analysis process.

ask the user to rate the application after a specific time. S1 explained the underlying reason: “We have a five star product in the AppStore. Together with the price this creates trust among the users.” Second, several users write about specific applications in their blogs or other social media. Both S2 and S5 reported on cases where a user had requested a feature which was later implemented, whereupon the user publicly praised the company. The subjects perceived such user-generated experience reports as important multipliers.

**Hypothesis 3.8.** *User feedback helps to advertise and market a product by conveying trust in the form of positive ratings and experience.*

### 3.3.2 Analysis

The interviews show that studied companies examine user feedback in order to create corresponding todos and prioritize them. We found that all studied companies follow a common, iterative process to analyze user feedback, which comprises two phases and six steps, as shown in Figure 3.2.

**Preparation Phase** The preparation phase comprises steps to collect feedback from different sources and assign it a coarse-grained category.

In the *first* step, developers merge the information they get over different feedback channels. This is necessary, since user feedback is obtained in various ways such as via email, phone calls, or over the AppStore. However, not all channels are equally suitable for a specific feedback type. S2 explained that in particular error reports were not really helpful when reported for instance by email, since typically important context information is missing: “In such a case we answer them to report the error via the integrated feedback mechanism.”

In the *second* step, developers read user messages and extract the included suggestions, which might be multiple for a single message. To what extent users provide digest-like feedback seems to depend particularly on the specific user audience, as claimed by S5: "In our market segment it is quite typical that users are willing to provide feedback. Sometimes our users see themselves more as testers or co-developers."

In the *third* step, developers decide if the feedback reports a problem or requests a feature. All studied companies make this simple distinction, in particular to be able to direct the report quickly in the right workflow. Error reports are typically more critical than feature requests, and therefore require a faster reaction. C1 and C2 explicitly maintain two feedback lists for this purpose. The other subjects did not report about explicit lists, but consistently made this distinction when talking about their workflows.

**Triage Phase** The triage phase comprises steps to assign feedback a priority and convert it into common project artifacts.

In the *fourth* step, developers assess the individual priority of the feedback. In the case of an error report, they estimate how critical a reported error is, while for a feature request developers assess if it is qualified, suitable to improve the product, and fits into the product roadmap.

In the *fifth* step, developers estimate the impact of the user feedback by investigating how frequently it occurs. To this end, they relate multiple pieces of feedback to each other to find commonly reported issues and needs. Our interviews suggest that this step is more difficult for feature requests than for error reports because of three reasons. First, in case of an error all studied companies could access stack traces, which could be symbolized and compared by tools. Feature requests on the other hand, are hand-written and need to be read and understood in order to compare them. Second, error reports typically refer to a singular, concrete event, while features and the underlying user needs might be complex. Third, while the similarity of error reports can typically be calculated by comparing stack frames, the similarity of feature requests might be rather subtle. S5 explained that two similar feature requests might seem unrelated at first glance: "Users might request different things, but their underlying goal might be the same. Our task is to abstract from the actual request, to think about the why, and to anticipate what is really missing on a conceptual level."

In the *last* step, developers establish a connection with conventional development tools and workflows. Our interviewees illustrated two particular ways. First, user feedback is discussed in team meetings. In the case of C1, C4, and C5, multiple developers collect user feedback and discuss the results in recurring meetings. We found that such exchanges were considered particularly useful as an additional tool to identify frequently occurring errors and feature requests. Second, developers create tickets from user feedback and insert them into the

Table 3.4: Perceived complexity of and satisfaction with current user feedback analysis practice.

| Subject | Amount of feedback | Perceived complexity of comparing feedback | Perceived satisfaction |
|---|---|---|---|
| S1 | high | somewhat difficult | somewhat unsatisfied |
| S2 | high | very difficult | very unsatisfied |
| S3 | low | somewhat difficult | somewhat satisfied |
| S4 | high | somewhat easy | undecided |
| S5 | medium | very difficult | undecided |

internal issue tracker. Interestingly, our impression was that user feedback is considered rather "fluid" until it manifests as a concrete task in such a tool.

**Hypothesis 3.9.** *Developers analyze user feedback in order to create prioritized tasks which fit into the product roadmap. The priority of a specific feedback depends on the frequency of its occurrence.*

### 3.3.3 Problems

All studied subjects agreed that it takes the most time to read and understand user feedback as well as to assess its impact by identifying for how many users it applies. We identified three main reasons: content and quality of user feedback, employed analysis techniques and feedback quantity, and the suboptimal communication between developers and users.

**Content and Quality of Feedback** Our interviews show that especially user feedback written in natural language is a problem for developers. These texts are typically written from a subjective perspective which requires developers to get into the user's mind to be able to reproduce her issue or request. But often information which would be essential to understand the feedback is missing. As illustrated by S1, users also typically regard their feedback as very critical while it might not be for the development team: "Some users are in the habit of shouting quite loud, but they don't really mean it." Moreover, depending on the software, features and user workflows can be complex leading to complex feature requests or issues. All subjects but S5 further explained that user feedback often has poor quality, in particular in the case of comments in the AppStore. According to them, a substantial part of these comments are unqualified and do not add any value. However, developers can only interpret feedback by reading or at least browsing it. It seems that the time required for this task increases developers' disenchantment with user feedback. S2 confirmed this interpretation: "Often users write a lot of text, but then it turns out that they just used or configured the software wrongly. At first this causes a lot of trouble, and then you find that it was the users' fault." Last, our interviewees reported that user feedback is often contradictory. For instance, S1 illustrated that part of their users consistently requests advanced, professional features, while the other part appreciates simplicity and the easy entry. Among our subjects contradicting preferences are currently ignored, and the internal product roadmap is followed instead, as S1 explained: "We ignore contradicting feedback until there is a clear opinion in the community."

**Hypothesis 3.10.** *Content and quality of user feedback affect its analysis. Natural language content and low feedback quality, as well as contradictory user feedback constitute particular problems for developers.*

**Manual Analysis and Quantity of Feedback** We found that the subjects analyze user feedback almost exclusively in a manual way. Consequently, in particular companies which receive a large amount of feedback spend considerable effort on its analysis. S2 pinpoints the limits of this practice: "Oftentimes we need to mark the mails as read, because we do not have the resources to really read them all." The analysis process suggests that developers even read a specific feedback multiple times while analyzing it. Our subjects reported that in particular assessing criticality and impact is a difficult and time-consuming task. To accomplish it, developers need to *compare* new feedback to the already reported in order to find duplicates or similar suggestions. Consequently, a single request can require the developer to examine multiple other feedback messages. We asked our subjects about their satisfaction with the currently established analysis process, on a 5-point Likert scale (1: *very unsatisfied*, 3: *undecided*, 5: *very satisfied*). The average response lies between *somewhat unsatisfied* and *undecided* (mean=2.6), as shown in Table 3.4. Although the gathered data does not allow for generalization, our interviews suggest that the amount of received feedback as well as the perceived complexity of the analysis task might have an influence on developers' satisfaction with current practice.

In contrast to other feedback types, automatically generated crash reports which contain machine-readable information, might be analyzed automatically. C1 and C5 employ analysis tools which are capable to use this information to group multiple reports according to the similarity of the reported stack frames, and thus to provide a measure for the impact of an error.

**Hypothesis 3.11.** *Developers analyze user feedback mainly manually and read single feedback multiple times.*

**Hypothesis 3.12.** *Developers spend most efforts on assessing the priority and impact of user feedback. The main reason therefore is that developers need to manually estimate how many users are affected by a specific feedback.*

**Communication Gap between Developers and Users** Our interviews depict a communication gap between developers and users. Because developers obtain user feedback over different channels, it often gets copied from one medium to another. This typically removes the possibility – if any – to react on the feedback without major efforts, for instance to ask clarification questions. Furthermore, it increases developers' distance to the reporter. Many feedback channels allow only for one-way communication from users to developers in any case, which is typically not effective. A prominent example often referred to by our subjects is the AppStore, where users publish feedback under a self-assigned name. While there are no built-in means to reach a specific user, S3 exemplified how developers try to bypass this gap: "We do 'social reverse-engineering', meaning that we try to find a user on Facebook or Xing who left a bad comment in the AppStore, by searching for the reported username. Then we ask if we could set up a remote debugging session." The absence of mechanisms which allow developers to contact feedback authors is perceived as a serious limitation by our subjects. S1 illustrated one effect of this gap: "Often there is no real error, but the user did not understand a specific feature. But we cannot get back to them. Instead we are forced to change the AppStore description to clarify the feature, or to add additional help files." Most feedback reporting mechanisms further do not allow users to edit their feedback. Subject S2 mentioned this as a severe limitation because of two reasons. First, users seem to report problems often too hastily, but later cannot cancel their feedback. Second, in some cases users provide clarifying information later, but cannot associate it with their prior feedback.

**Hypothesis 3.13.** *Users and developers are disconnected due to communication gaps in user feedback channels.*

## 3.4 User Involvement Requirements

We explored requirements for tools that support the user involvement workflow, asking subjects if and how user feedback should be consolidated and whether an assessment of its potential was regarded as helpful.

### 3.4.1 Tool Support

Our interviews show that developers generally would embrace tool support to consolidate user feedback. In particular they hope for a better structure among the gathered feedback and expect to save time, if user feedback could be grouped semi-automatically or automatically to support impact analysis. S4 pointed out that user feedback should be regarded as important development artifact, for which longitudinal tracking facilities are necessary. We think that companies receiving large amounts of user feedback will benefit from tool supported feedback consolidation, while we expect less impact on companies which only receive few feedback messages. Further, S1 and S5 remarked that new tools really need to add value to be accepted.

**Hypothesis 3.14.** *Developers need tool support to consolidate, structure, analyze, and track user feedback, particularly when feedback volume is high.*

## 3.4.2 Consolidation

We asked our subjects about their preferences for two different ways to consolidate user feedback: A *re-active* approach collects user feedback as before and analyzes the collected data afterwards. A *pro-active* approach tries to avoid duplicate feedback by presenting reporters relevant existing feedback they should vote for.

All subjects, except S5, preferred the pro-active approach and gave two main reasons. First, it takes most of the work off the developers' shoulders, since priority and impact are basically results of the number of votes. Second, it avoids duplicates and thus unnecessary traffic. Consequently it allows developers to concentrate only on important feedback.

S5 preferred the re-active approach to avoid biasing their users: "If you bias the users with other user feedback, you will probably restrict the creativity of the users." S5 acknowledged that the pro-active approach might be more appropriate in cases where developers receive a higher amount of feedback or users are not professionals. Interestingly, S4 regarded both approaches as reasonable for different situations, depending on privacy. According to the subject, the re-active approach is more appropriate for feedback which is currently not public such as phone calls, while the pro-active approach could be employed for publicly visible feedback such as AppStore reviews.

When asked according to what a tool should consolidate user feedback, studied subjects particularly favored three features. First, all subjects agreed that in particular *duplicates* should be grouped together. Moreover, subjects explained that while two users might have made the same *experience* with the software, their reports might be quite different, what should be considered by any such tool. Second, all subjects agreed that the *type of feedback* should determine a feedback group as before. Third, all subjects, except S4, reported that it would be helpful to know the *feature* to which a feedback applies. S1 mentioned that it would be helpful to be able to structure the received feedback "like the software".

**Hypothesis 3.15.** *To consolidate feedback, tools should group duplicate or similar feedback, capture the feedback type, and the feature for which it applies.*

**Hypothesis 3.16.** *Pro-active tools are appropriate for high volumes of non-confidential feedback, while re-active tools are more appropriate for less feedback and more professional end users.*

### 3.4.3 Assessment

Our interviews show that developers constantly need to assess the potential of user feedback to improve their software and its impact within the user community. All studied subjects confirmed that they would appreciate tool support for this assessment, because it may influence the further development plans. When asked how they would estimate the importance of user feedback, our subjects specified two main measures. *First* and foremost, the *frequency* of a specific feedback, i.e. how many users provided the same or a similar feedback. Most subjects utilize this quantitative benchmark already today, but in most cases they estimate it manually. S1 particularly regretted that they are therefore currently only able to react slowly on community trends: “We can only measure the frequency when we already have set our users up.” The main factors slowing down reaction time include the low visibility of existing feedback for other users, the scattering of feedback across multiple channels, as well as the manual analysis of user feedback by developers. *Second*, all subjects would include an *assessment of the individual user* who reported the feedback into the benchmark, and specified several measures for such an assessment. They considered it important to know for how long and how often the reporter has used the application, as pinpointed by S1: “If we have a user who uses our software twice a day for several hours, her feedback is probably more important than feedback of a casual user who ’accidentally’ bought the software.” S2 and S4 would further base their assessment of a user on the past experience with that user: “Frequency is important. But if a user always reports inappropriate feedback, we would not want this to be highly prioritized. Quality over quantity.” [S2]. All subjects stated that they manually keep track of particularly helpful users. However, only S2 and S3 also treat their feedback differently, which typically means reading it first. Moreover, all studied companies except C1 let these users know that they are important, for instance by thanking them via email (C3) or by issuing coupons (C3, C5).

**Hypothesis 3.17.** *To support impact analysis, tools should measure the frequency of user feedback and provide developers with an individual assessment of the reporter.*

## 3.5 Discussion

We discuss the implications of our findings for researchers, practitioners, and tool designers and summarize the limitations of our study.

### 3.5.1 Implications

User feedback contains important information for developers which helps to improve software quality and to identify missing features. In order to assess its relevance and potential impact, developers need to analyze the gathered feedback. High efforts are needed for this analysis, as developers mostly accomplish it manually.

#### 3.5.1.1 Implications for Researchers

In order to facilitate user feedback analysis, researchers should investigate the main factors that contribute to its complexity and explore how it can be automatized. We see two main implications. *First*, since user feedback is typically natural language text whose quality might be poor, it is difficult to determine its content automatically. Research should investigate how to extract information from such artifacts, for instance using microtext understanding or island parsing methods. *Second*, user feedback often lacks important context information which can facilitate understanding and help developers to reproduce reported issues. Consequently, researchers should examine how context information can be made available for the analysis process.

#### 3.5.1.2 Implications for Practitioners

Based on our results, we give three advices to developers. *First*, know your audience. Our study suggests that different user audiences provide feedback in different ways. Consumers seem to report ad-hoc, while professional users might elaborate more on their feedback. In either case, developers should provide suitable channels to gather the specific kind of feedback. *Second*, reduce the number of feedback channels. We found that feedback is typically scattered across several channels. As a consequence, developers merge feedback gathered over multiple channels, which reduces traceability and increases the gap to their users. Developers should identify which feedback type is supported best by which channel and discontinue other channels. Channels which allow for two-way communication should be preferred. *Third*, educate your users. Companies who decide to take user feedback seriously, need to explain their users how to provide helpful feedback. For instance, multiple requests within one message might complicate its analysis, while indicating the type of feedback might be helpful.

#### 3.5.1.3 Implications for Tool Designers

User feedback is a rich source of information. Our study has shown that developers work through this information in order to create conventional, prioritized development tasks. Developers need tool support to facilitate consolidation, structuring, analysis, and tracking of user's feedback, especially when the occurring volume is high. Tool designers should investigate how developers can be assisted during these tasks. Our results suggest that novel tools should identify similar and duplicate reports, capture the feedback type, and document the affected feature. Developers' main information needs include the impact of feedback in terms of its frequency as well as an assessment of the individual reporter, for instance how often and for how long she has used the software or how often she has already reported.

### 3.5.2 Limitations

As with any research methodology, our choice of research methods has limitations.

#### 3.5.2.1 Construct Validity

With this study we aimed at exploring problems and information needs in current user involvement practice. Construct validity therefore measures whether these concepts can be correctly reflected by means of interviews. First, interviews obviously rely on the statements of the participants, which might be subjective. While subjectivism is difficult to eliminate in interviews, we limit its effects by basing our findings exclusively on the statements of multiple subjects. Further, the semi-structured nature of our interviews allowed us to react on participants' statements, and to ask why-questions whenever needed, while guaranteeing at the same time that all participants answered the same questions. Second, subjects might not even be aware of the occurring problems. To limit this effect, we concretized questions related to problems, asking which tasks take much time, are subjectively difficult, and which information are needed to accomplish them. Other studies (e.g. [136, 162, 298]) support our findings, what makes us confident that the identified problems are real. Ensuring construct validity for empirical studies of software developers is always a complex task, specifically as such studies typically require the researcher to abstract from observed behavior or gathered information. Therefore, we encourage other researchers to replicate this study or enhance it for instance by means of observation.

#### 3.5.2.2 Internal Validity

Because our study is of exploratory nature, its internal validity is determined mainly by the evidence we have used to generate our hypotheses. We therefore discuss the two main factors which might affect the soundness of our observations, and illustrate how we tried to limit them. First, the interviewer might be biased towards the study proposition. In other words, he might have had a priori expectations and assumptions, and could have sought to confirm them. In order to limit this threat, we recorded the audio of each interview, transcribed the recorded audio, and sent the transcription back to the interviewees asking for corrections. Likewise, we sent the participants a copy of our hypotheses, and requested their feedback. All participants agreed with our findings. Second, participants might have given answers which are not completely reflecting their work practice, because they knew the results would be published. While this threat can never be completely eliminated in interviews, we addressed it by guaranteeing the complete anonymity of our participants and their companies.

#### 3.5.2.3 External Validity

The applicability of our findings has to be established carefully. The main limit to the generalizability of our findings results from the fact that we have interviewed only five subjects. We could increase confidence in our hypotheses by interviewing more subjects from a larger cross section of application domains and user audiences. On the other hand, all studied subjects are software professionals with over 3 years of practical experience in industrial companies and fill different roles. Moreover, studied projects span different domains, different amounts of users, and different user audiences, which makes us confident that our findings are representative. Finally, this study is of exploratory nature and was not designed to be largely generalizable. Its main idea is to explore and understand how developers deal with user feedback during software evolution, and which problems they encounter. To this end, we formulate hypotheses which should be validated by future studies of larger populations. Consequently, we avoid answering yes/no questions but concentrate on identifying common, real problems and information needs.

## 3.6 Related Work

Most studies about user involvement in practice explore the “early” phases of the software development lifecycle, i.e. mainly requirements engineering activities, and specifically investigate how and to which degree users are involved [284], and which effects such involvement has on product acceptance [165]. Only few other studies are concerned with exploring how developers work with user feedback during software *evolution*, and which problems they encounter.

Ko et al. [162] investigate what constrains evolution decisions in development teams, and found two main factors. First, developers were more likely to address feedback they believed to be shared by the majority of the users. Similarly, conflicting needs and preferences among the users reduced the probability for a feedback to be addressed. Our study confirms these findings (Hypotheses 3.9 and 3.10), and additionally concludes that developers often lack the necessary information to be able to assess the stake size for a given feedback (Hypothesis

3.12). The second factor is related to how deep an intervention would be required to address a specific user feedback. Correspondingly, our subjects reported that user feedback should fit into the product roadmap. Ko et al. conclude that feedback is a significant source of knowledge about user practices, what is confirmed by our results (Hypothesis 3.5). Interestingly, the authors argue that user feedback should be treated as a signal that further research is needed rather than as a guide for what to change. The main reason lies in the way developers currently react to user feedback which can lead to hardening the original software design. In contrast, we claim that developers need novel tools which fill their information needs, and allow them to measure the impact of user feedback.

Heiskari and Lehtola [136] investigate user involvement in practice without focusing on a specific development lifecycle phase, and identify several challenges which are confirmed by our study. First, similarly to our results the authors found that user information is scattered, unorganized, and difficult to access (Hypothesis 3.1), and that there is no clear and common process on understanding users (Hypothesis 3.4). Second, while feedback and other user information were considered important, the authors found that there is too little of this information available for developers (Hypotheses 3.5 and 3.12). Moreover, the study revealed that determining the average end user opinion is a hard task, what was confirmed by our interviews (Hypothesis 3.10). Finally, the authors discovered a need for the integration of user knowledge into existing development processes. We argue that this supports our finding that developers need tool support to deal with user feedback (Hypothesis 3.14).

Zimmermann et al. [298] specifically focus on developers' problems with bug reports in open source projects. One result of their study is that poorly written reports as well as missing information particularly hinder developers from understanding and reproducing issues, which is also confirmed by our interviews (Hypothesis 3.10). The authors further revealed a mismatch between information needed by developers' and information which users actually provide, what intensifies our Hypothesis 3.4: in the projects we studied, error reports typically include automatically generated information to support developers. Finally, the authors showed that well-known users' feedback is likely to get more attention, regardless of its importance. Similarly, our interviews showed that developers are interested in an individual assessment of the reporter (Hypothesis 3.17).

## 3.7 Summary

In this chapter, we described an empirical case study which we conducted with software professionals, in order to analyze the current practice of user involvement during software evolution, specifically for cases with large user audiences. We were particularly interested to find out how and why practitioners gather end user feedback during evolution activities. Our goal was to understand, what happens with user feedback in the development environment and why, and which information developers need. We mainly aimed at collecting problems and challenges in developers' workflows, but also tried to identify necessary characteristics of tools which could assist practitioners during their work with user feedback. Our findings can be summarized as below:

- Users provide feedback *frequently* and using different means, with the result that information is *scattered* across multiple channels in development environments, what complicates developers' work and widens the *gap* between users and developers. Users seem to know that their published feedback applies pressure on software companies, since they *intentionally* select more public feedback channels the more critical their issues are.

- Current user involvement practice is *not systematic.* There is no commonly agreed way neither how to *provide* nor how to *gather* user feedback, and *no guidelines* for users on what makes a "good" user feedback.

- Developers need user feedback, in order to assess if their product is *accepted* and to gather real-world *usage data.* User feedback is helpful to *improve software quality*, to *identify missing features*, and to *advertise* and *market a product.*

- Developers analyze user feedback in order to create prioritized *tasks* that fit into their roadmaps. The priority of these tasks mainly depends on the *impact* of the feedback, i.e. on the *frequency* of its occurrence. But for this purpose, developers need to assess how many users are affected.

- Several problems complicate the analysis of user feedback. Feedback messages are typically written in *natural language*, might have *poor quality*, and can *contradict* each other. Further, developers typically need to estimate feedback *impact* manually and consequently spend many efforts on this assessment, partly also because it involves reading single feedback multiple times. Finally, users and developers are typically *detached* since the utilized channels often only allow for one-way communication. As a result of these problems, developers often ignore feedback, simply sticking with their product roadmaps and development plans.

- Developers would embrace *tool support* to *consolidate*, *structure*, *analyze*, and *track* user feedback, in particular when feedback volume is high. Such tools should at least be able to group and count *duplicate* or *similar* feedback, capture the *feedback type*, and provide developers with an individual *assessment* of the *reporter.* In cases of high volume of non-confidential feedback *pro-active* tools are more appropriate, while less feedback and professional end users suggest that a *re-active* consolidation approach is suitable.

# Chapter 4

# Grounded Theory on Continuous User Involvement

«*Every act of knowing brings forth a world.*»

— Humberto R. Maturana and Francisco Varela, *Tree of Knowledge*

In the previous chapter we have analyzed how professional software developers deal with user feedback during software evolution and identified several problems. Developers need to identify similar reports and group them according to the user experience described, in order to assess the impact of user feedback. We found that these tasks require high reading, comprehension, and structuring efforts, in particular due to the high quantity and low quality of user feedback and because feedback is scattered across several channels.

Our goal is to facilitate developers' work by providing tool support for the consolidation of user feedback, thus lowering the required efforts for continuous user involvement. Specifically, we want to harness *user communities* to automatize the assessment of user feedback impact.

This chapter establishes the theoretical foundations of our approach by describing a grounded theory on continuous user involvement relying on user communities. To this end, we study two phenomena. First, we explore regularities in how users and developers communicate in open source communities in order to understand how these two groups should be connected. Second, we investigate how users currently provide feedback in application distribution platforms in order to understand how user feedback can be consolidated.

Grounded theory [70, 111, 192] is a systematic research methodology which was first described 1967 by the social scientists Glaser and Strauss. Since then it has been applied in different research areas mainly for qualitative research (e.g. [73, 75, 255]). Unlike traditional research methodologies, its goal is to derive a consistent set of hypotheses and discover a theory by analyzing the underlying empirical data. In the area of software engineering, such an approach

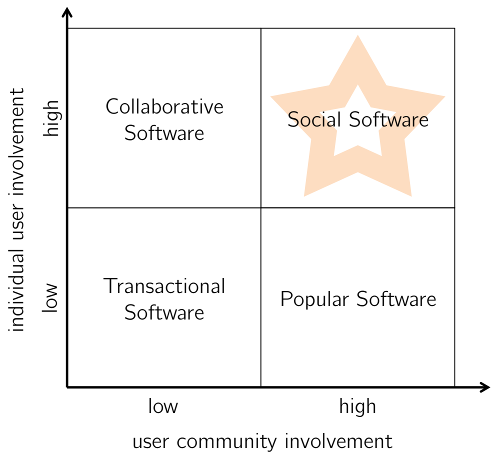


Figure 4.1: Benchmark for software socialness (following [173]).

is particularly appropriate to investigate social and human aspects – such as user involvement.

Section 4.1 motivates our research, while Section 4.2 presents our proposition. Section 4.3 briefly introduces the methodology we followed. In Section 4.4 we summarize how users and developers communicate in open source software communities. In Section 4.5 we present our findings on how users provide feedback in a large application distribution platform. Section 4.6 summarizes the grounded theory and concludes the chapter.

## 4.1 Motivation

The *socialness* of a software system is defined as the degree of involvement of its users and their communities in the software lifecycle [182]. In practice, socialness ranges from no involvement, to little involvement (e.g. public bug tracker), and to complete involvement in open source communities. Figure 4.1 illustrates four different classes of software systems according to their socialness. Users of *transactional software* are pure consumers with little to no possibility to contribute and promote the software. *Popular software* instead involves a large user community, which can be an indirect yet important means to create additional value. *Collaborative software* users are actively involved in the evolution by providing new ideas or performing other even more advanced engineering activities. Finally, *social software* involves a community of users, who actively contribute to the evolution of the software and pro-actively enlarge their community. Note that as opposed to collaborative software, *collaborative media* represent systems which facilitate collaboration (e.g. Wikis). Similarly, as opposed to social software, *social media* represent systems for social networking and interaction (e.g. Facebook or blogs). Software can be collaborative or social, independently from its features and domain.

The overall goal of our research is to increase the socialness of *any* software – be it closed source or not – since we assume that the continuous involvement of users and their communities is beneficial for both users and developers. We aim at making the involvement of users and communities a first order concern in software evolution by systematically utilizing valuable user experiences and volunteered resources.

## 4.2 Proposition

Our research proposition was twofold. First, we hypothesized that *user communities* bear high potential as indicator for the impact of user feedback. We assumed that such an indicator can help to reduce developers' efforts with continuous user involvement during software evolution. Further, we were convinced that users are willing and able to provide helpful feedback when chances are that it improves the software they use.

Second, we expected that conventional *user feedback* includes important information for developers, and that this information is occurring repeatedly. We assumed that such regularities can be exploited in order to facilitate the consolidation of user feedback. Moreover, we hypothesized that users influence the market with their feedback and establish latent communities even when the software is not open source.

## 4.3 Methodology

To collect qualitative and quantitative data which helps to test our proposition and investigate the current practical situation, we conducted two exploratory studies about user involvement with two different, complementary purposes.

First, we investigated how users and developers interact in a social environment, namely open source communities, where both user and community involvement are high. The goal of this study was to understand how these two groups communicate "in the wild", in order to draw conclusions on how they might be interweaved to increase software socialness in other, more controlled situations. The open source software industry is often regarded as the most popular and successful implementation of the open innovation paradigm [282, 65], where no boundaries between users and developers exist [237, 285]. Instead of following a strict product roadmap, an open source project can be seen as a genetic process: Source code (the genetic material) is transformed (mutated) by developers, while the community (evolution) decides which changes are successful and will survive [50]. Open source communities mainly rely on social media for their communication. Consequently, in the first part of constructing our grounded theory, we examined blogs in four large open source communities in an exploratory study. The results of this exploratory study have been published in [221]. For the sake of brevity, we decided not to include the complete study in this dissertation, but to summarize the obtained results (Section 4.4).

Second, we explored how users provide feedback in a more transactional, closed source environment, namely application distribution platforms, which are one of the most popular channels for users to provide feedback on software, as we have seen in the last chapter. Such platforms allow users to rate and review their applications. The resulting, publicly visible information can apply pressure on software companies, but reacting to it typically requires high efforts. As our goal is to facilitate developers' work with user feedback, we had to analyze the data for regularities which would allow for an automated or semi-automated consolidation. Consequently, in the second part of the grounded theory, we examined user feedback in one of the main application distribution platforms in a second exploratory study (Section 4.5).

## 4.4 Exploratory Study on User Involvement in Open Source Communities

Social media enable the creation and exchange of user-generated content [157]. Individuals can use them to interact with, share information with, and meet other individuals presumably with similar interests, forming large data, knowledge, and user bases. In recent years the number of users and use-cases of social media has grown rapidly [272]. For example, Facebook recently announced that it has over 1 billion active users – more than the whole population of the Americas and the Caribbean[1]. The usage of Facebook, Blogger & Co. is no longer limited to private scenarios such as finding school friends, sharing photos, or keeping a vacation diary. Professionals use more and more social media e.g. to organize a conference, market a new product, or coordinate an open source project.

The software engineering community has also recognized the potentials of social media to improve communication and collaboration in software projects [23]. For example, several studies have shown the role of Wikis for managing software documentation and collaboration [183]. Other authors suggested the integration of social media into development environments [123, 273, 278]. However, there exists no framework on the use of social media in software engineering. This study takes a step towards such a framework by exploring the role of blogs as a popular social medium in open source communities.

[1]http://www.facebook.com/press/info.php?statistics

To investigate how open source communities currently use social media, we divided the blogging ecosystem in active software developers and other stakeholders (including the users). We were particularly interested in understanding how and why these groups use blogs, and how their blogging activities are related to other project activities. From the results we drew conclusions on how current communication means between developers and other stakeholders can be improved (e.g. by revising software tools and processes), and how users and other stakeholders can be stronger involved in the development lifecycle.

In the remainder of this section, we briefly introduce the research questions (Section 4.4.1), present the applied research methods and the studied data (Section 4.4.2), and summarize the results we have obtained (Section 4.4.3). For the complete study, we refer the reader to [221].

### 4.4.1 Research Questions

The goal of this study was to understand how and why blogs are currently used by users and developers in a software community. We focused on three aspects: the actual *usage* of blogs in software projects, the *content* of these blogs, and the *integration* of blogging activities into the development workflows.

**RQ 3.(a) Blog usage** describes *how* software development communities "blog" (i.e. share information in blogs). For that, we analyzed the publishing frequency as well as the structure of blog posts, answering the following questions:

- *Publishing frequency*: How often do community members blog?
- *Post structure*: What are typical elements of a blog post and how often are they included or referenced?

**RQ 3.(b) Blog content** describes the *information* published in the blogs. This includes identifying topics (i.e. semantic entities) and their frequencies. In particular we answered the following questions:

- *Topics*: Which topics are discussed in blogs of development communities?
- *Topic popularity*: How popular are these topics (i.e. frequency distribution) across different communities?

**RQ 3.(c) Blog integration** describes *how* blogging activities are *integrated* in the development workflows. We examined usage patterns and content dependencies between blogs, source code repositories, and release repositories, answering the following questions:

- *Publishing patterns*: Are there particular patterns, which describe when blogs are used in the communities? In particular:

Table 4.1: Overview of research data

| | Eclipse | GNOME | PostgreSQL | Python |
|---|---|---|---|---|
| # posts | 10,333 | 18,323 | 3,385 | 18,660 |
| # bloggers | 328 | 342 | 112 | 405 |
| # commits | 239,659 | 252,831 | 30,745 | 45,116 |
| # committers | 467 | 2,294 | 34 | 178 |
| # blogging committers | 93 | 250 | 12 | 34 |

  – Release dependency: When are blogs posted in relationship to the software releases?
  – Activity dependency: When are blogs posted in relationship to particular development activities?

- *Published Information*: Are there relationships between the work performed and the information blogged? In particular:
  – Content dependency: Are blog post topics and particular development activities related?
  – Time dependency: To which degree are work performed and information blogged related in terms of time?

When answering these questions we distinguished between active developers (committers) and other bloggers. This allowed us to compare the behavior of developers and other stakeholders – including the users – in the studied communities.

### 4.4.2 Research Method and Data

To analyze the *usage* of blogs we applied descriptive statistics (for frequency calculation) and regular expressions (for analyzing the blog structure). We also conducted statistical tests to exclude the hazard factors and report on the error rates. To analyze the blog *content* and included information we used the Latent Dirichlet Allocation (LDA) topic modeling technique [32]. When applied to blog posts, this technique extracts keywords which belong together and groups them as topics. Finally, to study the *integration* aspect we ordered commit messages and blog posts as well as commit messages and releases by time and investigated the resulting stream of events. Thereby we looked for patterns and regularities using Sequential Pattern Mining [5].

Table 4.1 shows an overview of the studied data. We explored the behavior of over 1,100 bloggers in four large open source communities: Eclipse, GNOME, PostgreSQL, and Python. To this end, we analyzed over 50,000 blog posts and more than 568,000 commit messages.

## 4.4.3 Results

In all studied open source communities we observed regular and frequent blogging activities since several years and across many releases. This is not surprising, as blogs became one of the most popular social media for sharing and accessing software engineering knowledge in the last years [226]. While individual members only blog occasionally, the community as a whole constantly shares information and produces an average of up to six blog posts per day. These posts are written equally by committers and other community members.

Our results show that the studied communities use blogs as documentation tool to share knowledge and experiences, and to socialize and maintain community structures. A major finding of this study is that committers and other members blog on a high level of abstraction, e.g. frequently about features and domain concepts. At first glance this is surprising, as we expected developers to blog about models, technical abstractions, and source code related concepts. However, the public, social, and rather informal nature of blogs can be one of the reasons behind this information granularity. Blogs enable developers to document features, dependencies, known issues, and qualities of new releases in an informal and time-ordered way and to a broad audience. Studies showed that developers describe their work in short but regular commit messages [180]. Blog posts on the other hand are less frequent than commit messages, but comprise significantly more content. They rarely include source code but frequently high-level information and images. Therefore, blog posts seem to have rather the character of short documentations and tutorials.

Unexpectedly, we found that non-committers blog about more technical topics than committers in all studied communities. We think that this is due to the framework nature of the studied projects. Non-committers use Eclipse, GNOME, PostgreSQL and Python as infrastructure for their own projects. In their blogs, they frequently reflect their technical experiences, share code examples, patterns to solve particular engineering tasks, and howtos. This shows the importance of social media for making end users an integral part of software projects, enabling them to share their user experience and helping to create and maintain project knowledge [182].

Our results show that committers and other stakeholders communicate bidirectionally through social media. Committers provide development related information to other stakeholders, while the latter report about their experiences with using the software. Blog posts by committers frequently contain information about recent activities described shortly before in previous commit messages. Specifically, we found that developers post more often after corrective engineering than after forward engineering or re-engineering tasks. One silent implication of this finding is that bug fixes represent important information which should be shared with all stakeholders in a software community. Communicating these corrective actions to the community might have two implications.

First, developers publicly show their personal contributions and merits – an important social and motivational factor. Second, solved issues – which might have been reported by end users – indicate a healthy project and a healthy, social relationship between developers and users.

Likewise, blog posts about release announces and release plans make the community aware of the overall project status. We found that in particular non-committers publish most blog posts shortly after a new release, reporting about their experiences with the software. Again, this indicates that non-committers, including the actual end users, are continuously involved throughout the software lifecycle.

## 4.5 Exploratory Study on User Feedback in Application Distribution Platforms

Application distribution platforms such as Apple's AppStore[2], Google's Play[3], and Microsoft's Windows Phone Store[4] allow users to buy and deploy software applications with one click, while completely hiding the complexity of money transfer and liability for both users and developers. As a consequence, they are growing at high speed. As of September 2012, over 700,000 applications are available in the Apple AppStore, more than 500,000 in Google Play, and over 100,000 in the Windows Phone Store. The astronomic download numbers (around 2 billion per month in the Apple AppStore) make these platforms very attractive for developers.

The main use case of application distribution platforms is to allow users to download and install software. But in addition, they allow users to rate and review the offered applications. Users who have bought an application can rate it by assigning it a number of stars and publish a review about the software. Both ratings and reviews are public and visible to all users – and all developers.

The original purpose of this rating and review system is twofold. First, it ensures a user-oriented *quality* among the applications, since applications with higher ratings rank higher in so-called "top lists", which in turn increase the applications' publicity and thus its download numbers. This viral marketing is one of the main success principles of application distribution platforms. Second, it enables a user-based *recommendation* of applications, as it is commonly used in traditional shopping platforms, such as Amazon[5]. Since the providers of application distribution platforms benefit from the generated revenue, they are interested to sell as many applications as possible. Application recommendation is consequently the second main success principle of these platforms.

[2]https://itunes.apple.com/us/genre/ios/id36?mt=8
[3]https://play.google.com/store/apps
[4]http://www.windowsphone.com/en-us/store
[5]http://amazon.com

In Chapter 3 we described how users "abuse" the rating and reviewing feature to provide information about applications to its developers. Specifically, they seem to use this channel to publish error reports, feature requests, and other feedback on existing application features. While we found that this user feedback is important to developers, they cannot benefit from it in practice, since its analysis has to be done manually and therefore requires high efforts. The overall goal of this dissertation is to facilitate this part of developers' work by providing tool support for the *consolidation* of user feedback. The aim of this study was therefore to explore how users provide feedback in application distribution platforms and to analyze the data for regularities which would allow for an automated or semi-automated consolidation.

The remainder of the study is structured as follows. Section 4.5.1 introduces the research questions, research data, and methodology used. Section 4.5.2 summarizes our findings on the usage of feedback, the information included, and the impact of feedback on software market and user communities. Section 4.5.3 discusses the limitations of our study. Section 4.5.4 surveys related work.

### 4.5.1 Study Setting

We first formulate the questions that this study will answer. Then, we describe the overall method we used to collect and analyze the data. Finally, we present the actual data sets collected to perform our analysis.

#### 4.5.1.1 Research Questions

The goal of this study was to understand how and why users give feedback in application distribution platforms. We focused on three aspects: the actual *usage* of feedback by the end users, the *content* of this feedback, and the *impact* of user feedback on the user communities and software companies.

**RQ 4.(a) Feedback usage** describes *how* software users provide feedback. For that, we analyzed the feedback frequency as well as the meta-data of user feedback, answering the following questions:

- *Feedback frequency*: How often do users provide feedback?
- *Feedback meta-data*: On average, how long is user feedback, how are app ratings distributed, and how helpful is feedback for other users?

**RQ 4.(b) Feedback content** describes the *information* provided in the feedback. This includes identifying semantic entities and their frequencies. In particular we answer the following questions:

- *Feedback type*: Which different types of feedback do users provide?

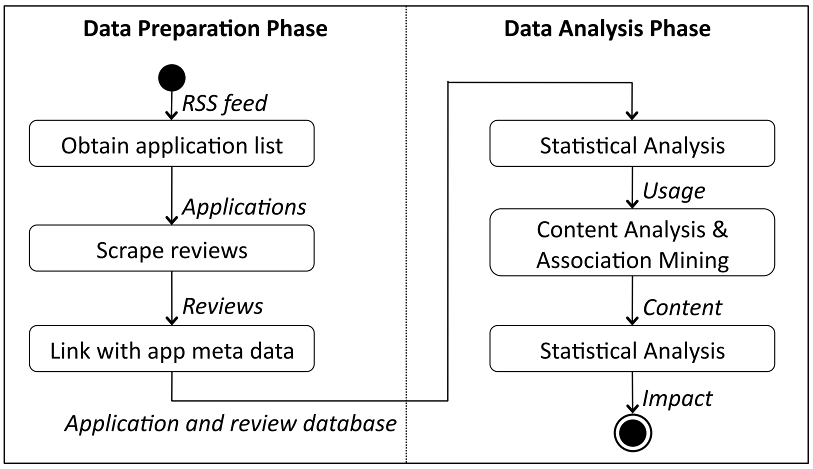


Figure 4.2: Research method.

- *Feedback patterns*: Are there recurring patterns in the feedback?

**RQ 4.(c) Feedback impact** describes how user feedback *influences* (i) the rating of an application and (ii) the user community. We examined the impact of user feedback answering the following questions:

- *Market impact*: Are there regularities in which user feedback accounts for a specific rating? In particular:
  - Feedback type and rating: Are specific feedback types related to particular ratings?
  - Feedback patterns and rating: Do particular feedback patterns correspond to specific ratings?
- *Community impact*: Are there any factors that influence the helpfulness of user feedback for other users? In particular:
  - Feedback helpfulness: Is feedback with a particular length more helpful than others? Are specific feedback types related to the perceived helpfulness?
  - Rating helpfulness: Are specific ratings more helpful than others?

When answering these questions we distinguish between paid and free applications, which allows us to decide if the price of an application has an impact on the studied research questions. This is important to understand if and how the application price influences user feedback and ratings.

#### 4.5.1.2 Research Method

Our research method consisted of two phases: a data preparation and a data analysis phase, as depicted in Figure 4.2.

**4.5.1.2.1 Data Preparation Phase** In the data preparation phase, we collected the data and created a database from it. We started by investigating the three biggest application distribution platforms and the available data: Apple's AppStore, Google's Play, and Microsoft's Windows Phone Store. All three include comparable data, which we are interested in: information about applications and associated reviews written by users together with a rating. We decided for Apple's AppStore because of two reasons. First, we had prior experience with the technology and applications. Second, we found a possibility to receive the data automatically. In the AppStore, applications belong to one of 22 disjoint categories[6], which should help users to find applications more quickly. Further, applications – called "apps" – are distinguished by their price in *free* apps and *paid* apps. Because of these classifications, we decided to draw a stratified sample of the data, including an equal number of free and paid applications for each category.

On September 16, 2012, we queried a list of the top 25 free and paid applications in each category utilizing an RSS feed generator provided by Apple[7]. This list includes the most downloaded applications and is updated hourly. Next, we parsed the list to extract the identifiers of all included applications. We then used an open source scraping tool[8], which we had modified in advance for our purpose, to query the list of reviews by iterating through all application identifiers. As last step in the data preparation phase, we linked each application's meta data such as application name, release date, price etc. with the obtained list of reviews and inserted the result into a MySQL database.

**4.5.1.2.2 Data Analysis Phase** The data analysis phase consisted of three steps, which answer the usage, content, and impact question, respectively. To analyze the feedback *usage* we applied descriptive statistics. We also conducted statistical tests to exclude hazard factors and report on the error rates. To explore the feedback *content* we performed a manual content analysis of a random sample of our data set (for finding included topics) and applied an association mining method (for identifying latent patterns). Finally, to study the *impact* aspect, we combined feedback meta-data with content and again employed a statistical analysis of the resulting data. We detail on each of these analysis steps in the corresponding result section.

#### 4.5.1.3 Research Data

Table 4.2 shows an overview of our data set. In total we obtained 1,126,453 reviews from 1,100 applications (550 free, 550 paid). In the AppStore reviews

[6]"Newsstand" is rather a tag than a real category, since apps inside it belong to other categories as well. Consequently, we did not consider it to avoid duplicates in our data.

[7]http://itunes.apple.com/rss/

[8]https://github.com/oklahomaok/AppStoreReview

Table 4.2: Overview of user feedback data set by categories. $N = 1126453$.

| # | app category | # feedback free apps | # feedback paid apps | mean price | max price |
|---|---|---|---|---|---|
| 1 | Books | 23,962 | 8,641 | 2.43 | 9.99 |
| 2 | Business | 35,265 | 23,997 | 4.11 | 16.99 |
| 3 | Catalogs | 9,517 | 5,725 | 1.35 | 4.99 |
| 4 | Education | 16,628 | 16,577 | 1.99 | 3.99 |
| 5 | Entertainment | 45,761 | 38,201 | 1.47 | 4.99 |
| 6 | Finance | 36,182 | 15,259 | 2.99 | 14.99 |
| 7 | Food & Drink | 19,066 | 8,318 | 2.27 | 9.99 |
| 8 | Games | 38,923 | 43,602 | 1.43 | 6.99 |
| 9 | Health & Fitness | 32,845 | 29,657 | 2.39 | 7.99 |
| 10 | Lifestyle | 39,954 | 12,607 | 1.51 | 4.99 |
| 11 | Medical | 17,203 | 4,160 | 2.39 | 5.99 |
| 12 | Music | 42,001 | 32,218 | 2.91 | 7.99 |
| 13 | Navigation | 15,961 | 10,528 | 5.35 | 49.99 |
| 14 | News | 28,041 | 19,822 | 2.07 | 4.99 |
| 15 | Photo & Video | 37,786 | 31,770 | 1.67 | 4.99 |
| 16 | Productivity | 37,426 | 32,695 | 3.15 | 9.99 |
| 17 | Reference | 28,269 | 16,393 | 1.91 | 4.99 |
| 18 | Social Networking | 51,899 | 26,691 | 1.59 | 3.99 |
| 19 | Sports | 22,374 | 7,173 | 3.55 | 29.99 |
| 20 | Travel | 24,350 | 10,939 | 2.91 | 9.99 |
| 21 | Utilities | 46,984 | 45,021 | 1.51 | 3.99 |
| 22 | Weather | 20,709 | 15,353 | 2.87 | 9.99 |
| | | Σ=671,106 | Σ=455,347 | Ø=2.27 | max=49.99 |

are reset every time a new version of an application is released. Therefore, the reviews in our data set were exclusively issued after the last release of the associated application. Less than half of the reviews (518,041 or 45.99%) specified the application version the reviewing user's feedback applied to. We could not explain this clearly, but we hypothesize that users might be able to enter feedback via a browser or via the AppStore software. The latter has access to the currently installed version.

Similarly, some reviews did not explicitly specify the date the review was published. But since we obtained the feedback in the order it had appeared online, we could compensate for this by investigating the date of the reviews that were published directly before and after. Since the feedback date does only include the day, we were able to obtain the real values. The oldest feedback was entered on 10 July 2008. Our data set therefore spans more than 4 years.

Most reviews for free apps were written in the category "Social Networking" (51,889 – 7.73%), least in the category "Catalogs" (9,517 – 1.42%). Most paid apps reviews in our data set belong to the category "Utilities" (45,021 – 9.89%), least reviews were published in the category "Medical" (4,160 – 0.91%). The most expensive applications on average belong to the category "Navigation" ($5.35 mean), while applications in the category "Catalogs" are the most economic on average ($1.35 mean). Overall, the average paid application in our data set has a price of $2.27, while the maximum price is $49.99 and occurs in the category "Navigation". The average application price across our complete data set, i.e. including free apps, is $0.92.

## 4.5.2 Results

### 4.5.2.1 Feedback Usage

**4.5.2.1.1 Feedback Frequency** Our data contains 1,126,453 reviews from 918,433 distinct reviewers, making around 1.23 reviews per reviewer. Of these reviews, 671,106 apply to free apps, while 455,347 are written for paid apps. The difference between the resulting average (1220.19 reviews per free app and 827.90 reviews per paid app) is significant in our data set, so that we conclude that in total more feedback is written for free apps than for paid apps (two-sample t-test, $p<0.001$, CI=0.99). Most probably the difference results from the larger user number of free applications. In our data set we counted 568,599 distinct reviewers for free apps and 389,563 distinct authors of reviews for paid apps. Consequently, the user ratio of free to paid apps (1.46:1) matches quite well the review ratio of free to paid apps (1.47:1).

We first studied the number of reviews by individual reviewer. We found that 84,567 reviews (7.51%) were made by anonymous reviewers using the username "Anonymous". In addition, we observed several other anonymous usernames, such as "????" (353 reviews), "???" (330 reviews), and so forth. In total, 57 of

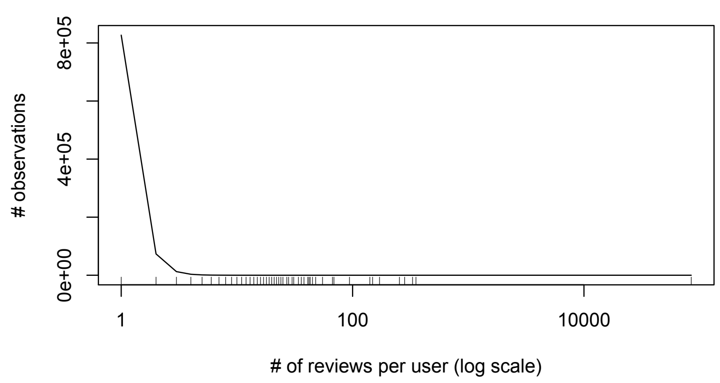


Figure 4.3: Distribution of reviews per user.

the top 1,000 users in our data are anonymous in this way, leading to a total of 87,282 anonymous reviews (7,75%). The remaining reviews are authored by users with non-anonymous usernames, even if it is questionable if and how they really identify a user. Overall, the issued reviews per user seem to follow a power-law distribution, as illustrated in Figure 4.3. 826,874 (90.03%) of the reviewers in our data set have written only 1 feedback. In contrast, only 1,183 (0.13%) account for more than 5 reviews.

On average users in our data set have provided 22.09 reviews per day per app. Again, more users provide feedback for free apps than for paid software. Concretely, we found 36.87 reviews per day for free applications, while only 7.18 daily reviews were issued for paid applications. Again, this difference is significant in our data (two-sample t-test, $p<0.001$). Figure 4.4 illustrates how the amount of daily feedback is distributed across different application prices.

Application users publish most feedback per day in the category “Games” (median 31.24), followed by “Social Networking” (median 8.82) and “Utilities” (median 8.75). We use medians rather than means to measure average daily feedback, since the distributions are positively skewed, as depicted in Figure 4.5. Least feedback is provided for applications in the categories “Catalogs” (median 0.30), “Medical” (median 0.34), and “Books” (median 0.53).

On application-granularity level, most feedback was given for the free application “Facebook”, which ranks 6th in the category “Social Networking”. For this application, users published 4,275 reviews in just one single day. The average of these ratings is 3.95 stars. The least feedback in our data set was provided for the application “Packers Radio & Live Scores” that ranks number 16 in the category “Sports” and costs $0.99. In this case, 2 users provided feedback in 303 days. Both of them gave 5 stars.

To understand if and how user feedback depends on time, we investigated users’ behavior after a new release. To this end, we first estimated for each feedback how many days after the first feedback on this application it had been provided. From these timespans, we calculated the distribution of the reviews

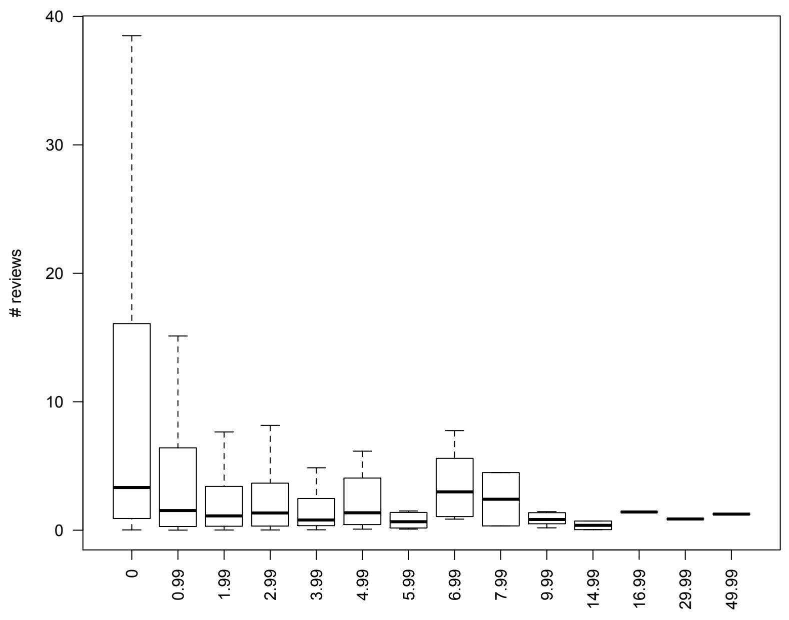


Figure 4.4: Daily feedback per app price.

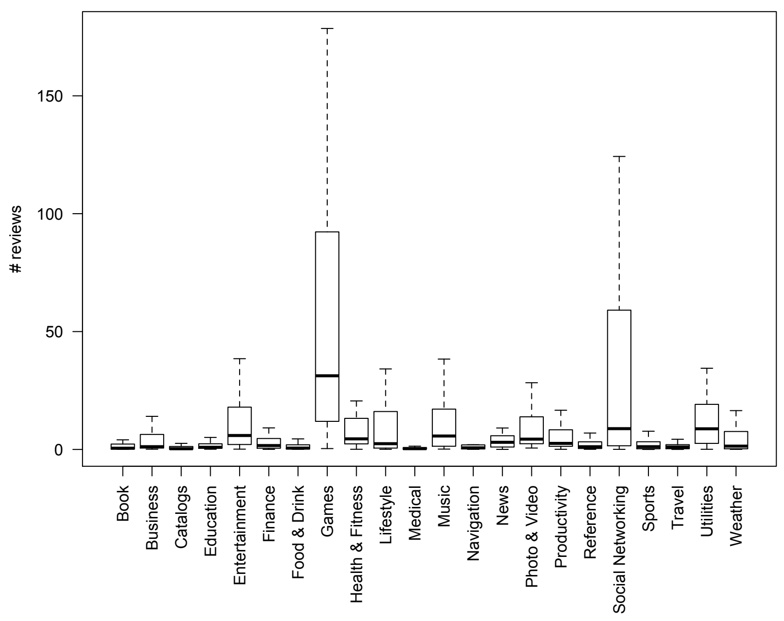


Figure 4.5: Daily feedback per app category.

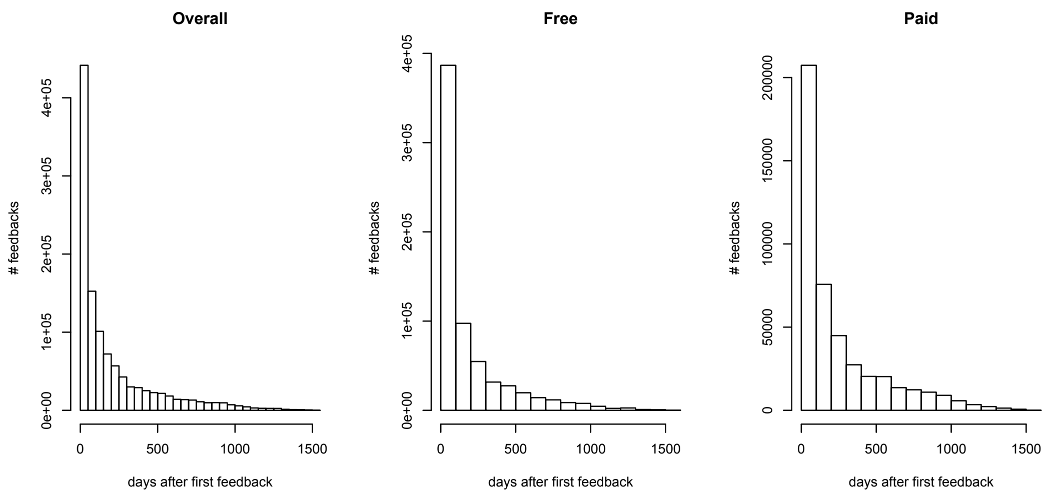


Figure 4.6: Relative distributions of feedback over time.

over time. As shown in Figure 4.6, users quickly give less feedback over time. Although the distribution is not exponential (a Kolmogorov-Smirnov test rejected this hypothesis with $p<0.001$), users give most feedback in the first few days after a release, leading to a long tail over time. This strongly suggests that user feedback is triggered by new releases.

#### 4.5.2.1.2 Feedback Meta-Data

We first studied the length of feedback in our data set. Overall, the feedback length ranges from 1 character to 6,000 characters in our data set. The median feedback length across all applications is 61 characters (106.09 mean). 2,802 (0.25%) reviews comprise only exactly 1 character, while 6 app reviews contain 6,000 characters. None of these very long texts contain usable information. Rather, it seems that the corresponding users randomly typed characters or repeated sequences of characters and white space to generate visual patterns. 863,951 (76.70%) of all feedback in our dataset are shorter than 140 characters, which is the length of a Twitter message. 905,974 (80.43%) contain less characters than an SMS text message (160). Over 99% of the feedback contains less than 675 characters, which corresponds to around a third of a printed page. We therefore conclude that application feedback are mostly short messages, more similar to a tweet than to other communication artifacts such as email.

As shown in Figure 4.7, feedback length seems to increase with application price. Although we could not directly find a significant linear correlation between app price and feedback length, we were able to show a significant increase in feedback length between lower-price and higher-price applications. To this end, we divided our data into two disjunct sets. The *lower-price* set includes

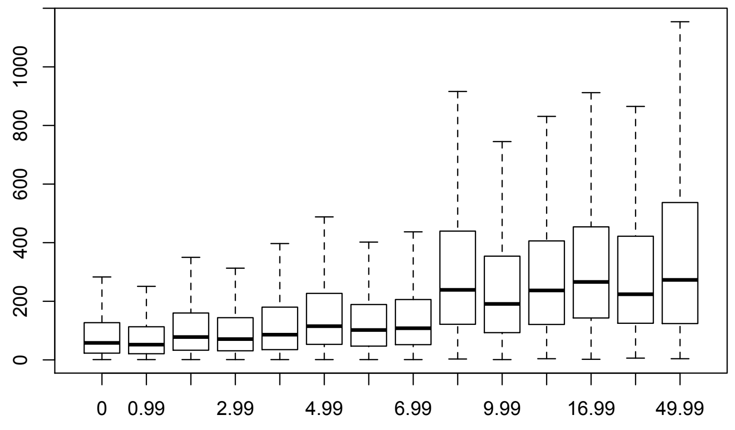


Figure 4.7: Feedback length by application price.

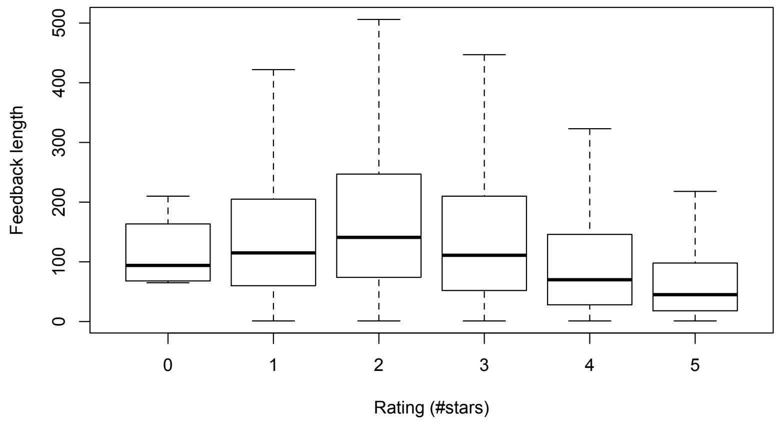


Figure 4.8: Feedback length and ratings.

applications with a price up to \$6.99, while the *higher-price* set contains all apps which are more expensive. A two-sample Wilcoxon rank sum test showed that users write significantly longer feedback for higher-price than for lower-price applications (p<0.001).

As shown in Figure 4.8, feedback length seems to be related to the rating in stars. By using a series of two-sample Wilcoxon rank sum tests, we could show the following relationships for the median length of feedback according to its rating: $m_1 < m_2 \wedge m_2 > m_3 \wedge m_3 > m_4 \wedge m_4 > m_5 \wedge m_1 > m_3$ (p<0.001). Our data set does not include enough samples of zero star ratings to make any assumptions. But since usually only 1–5 star ratings are allowed, we might exclude these reviews. We think that this result is quite interesting. One interpretation is that users tend to say less the more they like an application – a sign that there is less to improve. Conversely, it indicates that written feedback is mostly used for improvement requests.

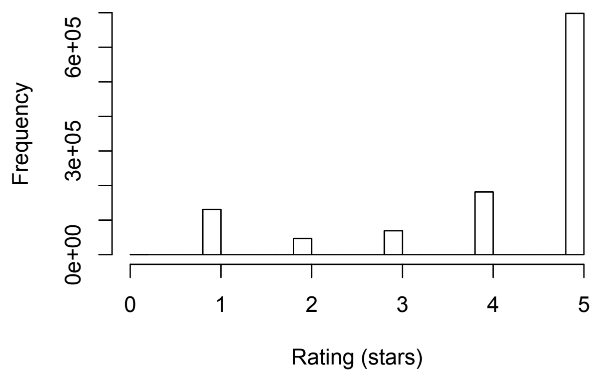


Figure 4.9: Frequency of different ratings.

Ratings in our data set are overall very positive, leading to an average rating of 4.13 stars. 697,932 (61.96%) of the reviews contain a 5 star rating, 879,373 (78.07%) give at least 4 stars. Only 130,940 (11.62%) reviews rate the application with 1 star – the lowest possible rating. Figure 4.9 shows an overview of the underlying frequency distribution. We obtained our data set by querying the top downloaded applications in each available category. Thus, although our data sample contains a large number of low ratings (177,887 or 15.79% reviews with less than 3 stars), we cannot exclude an influence of the applications' rank on the overall ratings (and obviously vice-versa). A $\chi^2$-test of independence confirmed the hypothesis that rank and ratings are not independent in our data ($\chi^2$=2,661,333, $p<0.001$).

In the AppStore, users may give a quality indicator for existing feedback by other users. They may do so by rating the "helpfulness" of a specific feedback. In our dataset, only 67,143 (5.96%) reviews are rated by other users regarding their helpfulness. From these, 38,519 (57.37%) are considered 100% helpful. Only 19,118 (28.47%) rated reviews are rated useless by more than half of the rating users. Interestingly, 16,671 (24.83%) are rated completely useless. This means that the user community agrees in over 82% of the cases about their opinion. To further investigate the distribution of helpfulness, we created the corresponding density plot (see Figure 4.10). The result suggests a bimodal or multimodal distribution, which we could confirm using Hartigans' dip test [127] ($p<0.001$). As it turns out, if feedback is rated, it is regarded either very helpful or very useless by the user community.

#### 4.5.2.2 Feedback Content

To investigate the content of user feedback in our data set, we employed an iterative content analysis technique. To this end, we first drew a stratified random sample of our data by randomly picking 12 reviews from each of the

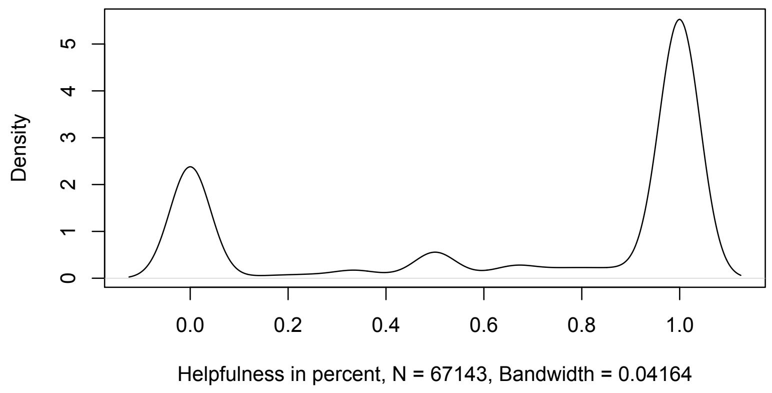


Figure 4.10: Density plot of feedback helpfulness as rated by other users.

22 application categories for both free and paid applications, leading to 528 reviews. Next, two researchers (including the author) independently from each other coded the random sample to identify topics included in the feedback. Since most feedback contained more than a single topic, we allowed the assignment of multiple topics. We obtained two sets of topics, a set $T_1$with 16 items and a set $T_2$ with 21 items. By discussing the identified topics, we discovered that $T_1$ was a strict subset of $T_2$, but that $T_2$ allowed more specific distinctions regarding certain information entities. We therefore decided to take $T_2$ as base for further coding, with one modification: We removed the 4 least frequent topics, as they had been found only in 1 or 2 reviews. The resulting 17 topics served as our coding guidelines for the rest of the content analysis.

Before the final coding round, we doubled the random sample to 1,100 reviews. This allows us to make predictions about our data set at the 95% confidence level, accepting an error margin of 3%. Both researchers then coded this larger sample, again independently from each other. Finally, we discussed feedback for which the assigned codes did not match and decided together for a final code.

**4.5.2.2.1 Feedback Type** The final results are shown in Table 4.3. We found that the most popular topic is "*praise*", which denotes any kind of praising the application under feedback. This topic is predominant in over 75% of the analyzed samples. The second most popular topic is "*helpfulness*", which describes a use case or situation where the application proved to be helpful to the user. It is predominant in more than 20% of the feedback. Further the topics "*feature information*" as well as "*shortcoming*" are predominant in over 13% of the analyzed feedback sample.

While reading the sample feedback, we made three interesting observations. First, the quality of feedback in our random sample varies quite strongly. On the one hand, there are app reviews with a high quality, which suggest interesting new features and justify their suggestions profoundly. On the other hand,

Table 4.3: Topics in user feedback. $N = 1,100$.

| # | topic | description | frequency |
|---|---|---|---|
| t1 | praise | expresses appreciation | 75.36% |
| t2 | helpfulness | scenario the app has proven helpful for | 22.45% |
| t3 | feature information | concrete feature or user interface | 14.45% |
| t4 | shortcoming | concrete aspect the user is not happy with | 13.27% |
| t5 | bug report | bug report or crash report | 10.00% |
| t6 | feature request | asks for missing feature | 6.91% |
| t7 | other app | references a different app, e.g. for comparison | 3.91% |
| t8 | recommendation | suggests acquisition | 3.82% |
| t9 | noise | meaningless information | 3.27% |
| t10 | dissuasion | advises against purchase | 3.27% |
| t11 | content request | asks for missing content | 2.91% |
| t12 | promise | expresses will to give better ratings under conditions | 2.00% |
| t13 | question | asks how to use specific feature | 1.27% |
| t14 | improvement request | requests improvement (e.g. because app is slow) | 1.18% |
| t15 | dispraise | opposite of praise | 1.18% |
| t16 | other feedback | references or answers other feedback | 1.09% |
| t17 | howto | explains other users how to use application | 0.91% |

some feedback does not add any value to the numeric rating of the application. Second, users tend to become insulting quickly. Especially when they have spent money they seem to forgive the developers nothing. Feedback like "Fire the idiot who designed this app!" is far from constructive and explains developers' disenchantment with user feedback. Third, users seem to complain frequently about removed or changed features. We see this as an indicator for users' getting used to specific workflows with an application. Changing an application in a way that affects users' workflows seems to be a source of dissatisfaction.

In order to estimate the diversity of information in user feedback, we calculated the distribution of number of topics per feedback. Overall, 6 (0.55%) reviews contained 5 topics, 22 (2.00%) reviews contained 4 topics, 3 topics were included in 116 (10.55%) reviews, 2 topics in 427 (38.82%) reviews, while 528 (48.00%) of the reviews contained only one topic. In other words, the majority of feedback (52%) contains more than one topic.

To further interpret and compare the information included in feedback, we grouped the resulting topics into the following themes:

1. **Community**. These topics represent community and social aspects. Specifically, references to *other feedback* and *other apps*, *questions* to other users, *howtos* explaining how to use the application, as well as *recommendations* and *dissuasions* are included in this theme.

Table 4.4: User feedback topic categories. $N = 1100$.

| category | topics | frequency |
|---|---|---|
| rating | t1, t12, t15 | 856 (77.82%) |
| user experience | t2, t3 | 359 (32.64%) |
| maintenance | t4, t5, t6, t11, t14 | 340 (30.91%) |
| community | t7, t8, t10, t13, t16, t17 | 146 (13.27%) |

2. **Maintenance**. This theme captures topics related to the improvement of an application. All requests – *feature*, *content*, and *improvement requests* – as well as *shortcomings* and naturally *bug reports* belong to this theme.

3. **Rating**. This theme contains topics that are related to a judgement of an application, particularly *praise* and *dispraise*, but also *promise*, which expresses the user's intention to change her judgement given certain improvements.

4. **User experience**. This theme comprises topics related to users' descriptions of the application in action. These are *helpfulness*, which captures use cases where the application has proven helpful, as well as *feature information*, which includes descriptions of application features and user interface.

Table 4.4 summarizes the identified themes together with the included topics, and shows their frequency across the random sample. Frequency denotes the number and percentage of feedback that contained at least one of the associated topics.

In our random sample, "*rating*" is by far the most frequent theme with a frequency of over 77%. This corresponds to the main intention behind application distribution platforms after the distribution itself, which is giving other users indicators for good applications and thus guaranteeing a high quality among the applications. The second most frequent theme is "*user experience*" which is predominant in nearly one third of all feedback. We think that the high popularity of this theme is interesting. It suggests that users tend to share their experiences with other users, presumably to justify their statements about an application such as ratings, recommendations, or dissuasion. The theme "*maintenance*" is predominant in around 30% of all feedback. This suggests that despite the overall quite positive ratings, users often externalize requests for improvement. Last, the theme "*community*" shows up in around 13% of the analyzed samples. We think that this number is quite high, given the original purpose of the AppStore as application distribution platform. It might suggest that users naturally tend to form communities, i.e. to react to each other, ask questions, or publish presumably helpful information.

Table 4.5: Frequent topic patterns in user feedback. $N = 1100$.

| # | pattern | support | avg. rating |
|---|---|---|---|
| p1 | {helpfulness, praise} | 22.18% | 4.86 |
| p2 | {feature information, praise} | 14.18% | 4.83 |
| p3 | {feature request, praise} | 4.64% | 4.37 |
| p4 | {helpfulness, feature information, praise} | 4.27% | 4.87 |
| *p5 | {helpfulness, feature information} | 4.27% | 4.87 |
| p6 | {praise, recommendation} | 3.73% | 4.90 |
| p7 | {other app, praise} | 2.64% | 4.79 |
| p8 | {praise, shortcoming} | 2.64% | 4.24 |
| p9 | {content request, praise} | 2.27% | 4.60 |
| p10 | {dissuasion, shortcoming} | 1.82% | 1.45 |
| p11 | {helpfulness, praise, recommendation} | 1.73% | 4.95 |
| *p12 | {helpfulness, recommendation} | 1.73% | 4.95 |
| p13 | {bug report, dissuasion} | 1.27% | 1.21 |
| p14 | {bug report, shortcoming} | 1.27% | 1.57 |
| p15 | {bug report, praise} | 1.18% | 4.23 |
| p16 | {feature information, praise, recommendation} | 1.09% | 4.83 |
| *p17 | {feature information, recommendation} | 1.09% | 4.83 |
| p18 | {improvement request, praise} | 1.00% | 4.18 |
| p19 | {feature information, other app, praise} | 1.00% | 4.91 |
| *p20 | {feature information, other app} | 1.00% | 4.91 |

**4.5.2.2.2 Feedback Patterns** We utilized the association mining algorithm ECLAT[9] by Zaki [295] to identify co-occurrences of topics in our data sample. Association mining (or association rule mining) [4] is a data mining method for discovering relationships between different variables based on their co-occurrence in databases. It takes as input a database containing at least two different variables as well as a parameter specifying the minimum support $\sigma$ for the relationships to discover. The output of association mining methods are frequent itemsets, i.e. sets of values which co-occur in at least $\sigma$ percent of the data.

Our goal was to find sets of topics which co-occur in our data set with a higher frequency than others. Consequently, we built a database which contained the topic codes for each feedback in our random sample. We ran the ECLAT algorithm with a minimum support of $\sigma = 0.01$ and a minimum pattern length of 2, which means that results should only include itemsets with at least 2 topics.

Table 4.5 shows the 20 patterns[10] which we obtained with these thresholds. The most frequent pattern is {helpfulness, praise}, which is present in more than

[9]Equivalence CLAss Transformation

[10]Patterns marked with an asterisk (*) are not closed, which means that there exists at least one super-pattern that has the same support as the pattern.

20% of all feedback. It describes the usefulness of an application together with a positive rating. A concrete example in our data set is "Great for uploading receipts on the go. Easier than reconciling on the computer." The second most frequent pattern {feature information, praise}, which applies to over 14% of our random sample is similar to the first, with the difference that it describes more concretely a positive feature or functionality of an application. A concrete example in our data set is "I love that this app takes less than ten seconds to let you know where your battery life is!!! I love it." The third pattern {feature request, praise} is predominant in nearly 5% of our random sample. It illustrates positive feedback which also includes a feature request. From such a pattern we would expect a lower rating of the application than we would for instance from the first pattern. To test this hypothesis, we investigate regularities between feedback content and ratings in the following section.

### 4.5.2.3 Feedback Impact

**4.5.2.3.1 Market Impact** To study the market impact of feedback, we explored relationships between application ratings and feedback topics as well as feedback patterns.

We first tested the independence of the identified topics and the final rating of the application by the user with a number of $\chi^2$-tests. The results show that the topics "*other app*" ($p=0.90$), "*other feedback*" ($p=0.69$), and "*howto*" ($p=0.56$) are independent from the rating, while all others are not ($p<0.05$).

To further study the impact of specific feedback types on the application rating, we calculated for each topic the distribution of the associated ratings as well as the average rating across all feedback where it occurs. Table 4.6 illustrates the results and relates them to the overall average rating of the feedback in our random sample. The most positive topic is "*recommendation*", followed by "*helpfulness*" and "*feature information*", while the most negative topic is "dissuasion", followed by "dispraise" and "bug report".

The results suggest two interesting findings in the light of the previously identified feedback themes. First, it allows to order the *maintenance* topics according to their impact on the user-specified rating as follows. *Content requests* (4.25 stars on average) are the least critical maintenance feedback. Their average rating even lies above the overall average rating of the random sample. *Improvement requests* (3.92 stars on average) and *feature requests* (3.89 stars on average) are more critical, but their average rating still lies above the theoretical middle of 3 stars. *Shortcomings* (2.10 stars on average) have a definite negative impact on feedback rating. *Bug reports* (1.84 stars on average) are most critical. Second, it suggests that *user experience* topics are mainly included in positive reviews. In other words, users do not tend to include their experience with an application in negative feedback. Given that such information is most important for corrective maintenance activities and in particular to fix bugs (see

Table 4.6: Distribution of ratings across topics in user feedback. $N = 1100$ – 🟥 1 star, □ 2 stars, ▨ 3 stars, ▩ 4 stars, ■ 5 stars.

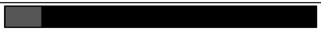
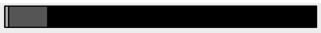
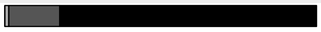
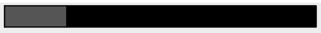
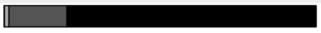
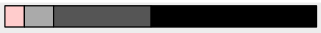
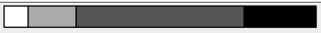
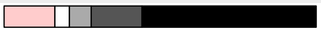
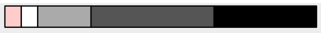
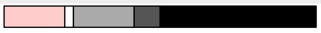
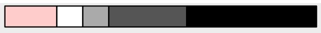
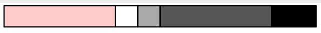
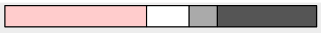
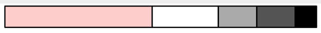
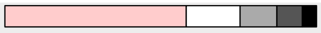
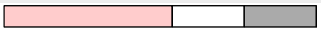
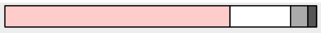
| # | topic | avg. rating | rating distribution |
|---|---|---|---|
| t8 | recommendation | 4.88 |  |
| t2 | helpfulness | 4.85 |  |
| t3 | feature information | 4.81 |  |
| t17 | howto | 4.80 |  |
| t1 | praise | 4.78 |  |
| t11 | content request | 4.25 |  |
| | *avg. sample rating* | *4.08* | |
| t14 | improvement request | 3.92 |  |
| t7 | other app | 3.91 |  |
| t6 | feature request | 3.89 |  |
| t9 | noise | 3.67 |  |
| t16 | other feedback | 3.67 |  |
| t13 | question | 2.86 |  |
| t12 | promise | 2.27 |  |
| t4 | shortcoming | 2.10 |  |
| t5 | bug report | 1.84 |  |
| t15 | dispraise | 1.69 |  |
| t10 | dissuasion | 1.39 |  |

Table 4.7: Top five topics per rating.

| # | 1 star ($N = 138$) | 2 stars ($N = 56$) | 3 stars ($N = 58$) | 4 stars ($N = 166$) | 5 stars ($N = 682$) |
|---|---|---|---|---|---|
| 1 | shortcoming (50.00%) | shortcoming (55.36%) | shortcoming (31.03%) | praise (92.77%) | praise (97.07%) |
| 2 | bug report (46.38%) | bug report (33.93%) | bug report (22.41%) | helpfulness (18.67%) | helpfulness (31.23%) |
| 3 | dissuasion (18.84%) | dissuasion (12.50%) | feature request (22.41%) | feature request (18.07%) | feature inform. (19.21%) |
| 4 | promise (7.25%) | feature request (7.14%) | praise (20.69%) | feature inform. (15.66%) | recommendation (5.43%) |
| 5 | other app (5.07%) | promise (5.36%) | noise (12.07%) | shortcoming (10.84%) | feature request (3.67%) |

e.g. [298]), this finding indicates that user feedback in this form will unlikely help developers to improve their applications.

To break the impact of topics on ratings further down, we calculated the top five topics per rating. Table 4.7 shows the results which confirm the two findings. *Shortcomings*, *bug reports*, and *feature requests* have high influence on negative ratings, while *helpfulness* and *feature information* have high impact in positive feedback.

Next, we investigated the relationship between feedback patterns and specific ratings. To this end, we calculated the average rating for each of the identified patterns. The results are included in Table 4.5 on page 78. As it turns out, the average ratings vary largely across the different patterns. The most positive (closed) pattern is {helpfulness, praise, recommendation}[11], which has an average rating of 4.95 stars. As we had hypothesized, the pattern {feature request, praise} has a lower average rating (4.37 stars). The pattern {bug report, dissuasion} accounts for the lowest average rating (1.21 stars), which corresponds to the negative message transmitted by reporting a bug and dissuading other users from buying the application.

**4.5.2.3.2 Community Impact** To study the impact of feedback on the user community, we investigate relations between feedback helpfulness and feedback length, content, as well as rating.

At first glance, we could not observe any linear or polynomial correlation between the feedback length and its helpfulness for other users. However, a $\chi^2$-test showed that feedback length and helpfulness rated by other users are statistically dependent ($\chi^2$=2043547, p<0.001). To further study this phenomenon, we split feedback into three groups according to its helpfulness. *Low helpfulness* includes feedback rated helpful only by up to 33% of the users who rated

[11]{helpfulness, recommendation} is a sub-pattern with equal support.

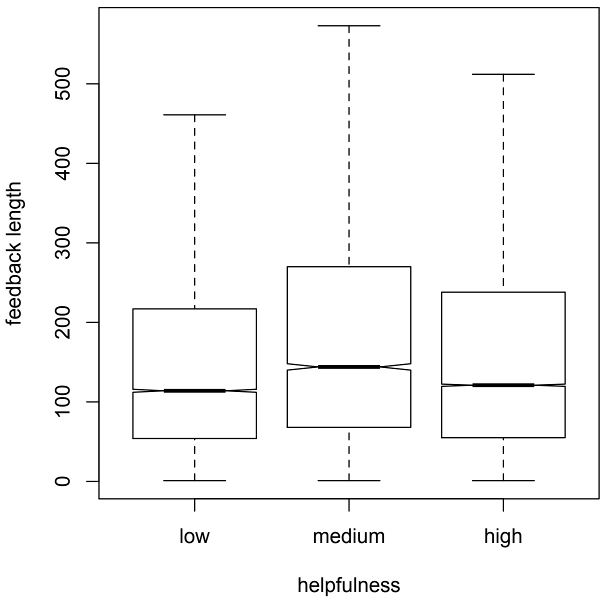


Figure 4.11: Helpfulness and feedback length. $N = 67143$.

it. *Medium helpfulness* indicates that 33–66% of the users who rated the feedback found it helpful. *High helpfulness* includes feedback considered helpful by most, i.e. more than 66%, of the users who rated it. Figure 4.11 shows how feedback length is distributed across these three categories. By using a series of two-sample Wilcoxon rank sum tests, we could show the following relationships for the median length of feedback according to its helpfulness: $m_{low} < m_{medium} > m_{high} \wedge m_{low} < m_{high}$ (p<0.001). This result means that feedback with low helpfulness is the shortest, while feedback with medium helpfulness is the longest in our data set. When going beyond the descriptive level, the significance of the length difference between low and high helpfulness feedback is open to dispute, as the median lengths of these categories differ only in 7 characters ($m_{low} = 114, m_{medium} = 144, m_{high} = 121$). In contrast, the difference to feedback with medium helpfulness lies between 23 (19.01%) and 30 (26.32%) characters.

What is remarkable, however, is the fact that all feedback whose helpfulness has been rated is significantly longer than other feedback. The 67,143 reviews which have been rated in our data set have a median length of 121 characters, while the median length of the remaining 1,059,310 reviews is 58 characters, which is *less than half*. This difference is significant in our data set (two-sample Wilcoxon rank sum test, p<0.001). This result suggests that longer feedback is more likely to be rated by other users. The reason might be that longer feedback contains more information to identify with or to dislike.

Next, we studied how feedback helpfulness and feedback content are related. Unfortunately only 74 (6.73%) reviews of the random sample which we had used

Table 4.8: Helpfulness of topics in user feedback. $N = 74$.

| # | topic | avg. helpfulness | N |
|---|---|---|---|
| t6 | feature request | 90.33% | 7 |
| t3 | feature information | 86.36% | 11 |
| t1 | praise | 83.93% | 51 |
| t16 | other feedback | 83.33% | 3 |
| t2 | helpfulness | 81.50% | 18 |
| t11 | content request | 75.00% | 4 |
| t14 | improvement request | 70.83% | 4 |
| t7 | other app | 67.33% | 7 |
| t17 | howto | 65.14% | 5 |
| t4 | shortcoming | 63.18% | 13 |
| t5 | bug report | 53.47% | 14 |
| t10 | dissuasion | 0.00% | 1 |
| t12 | promise | 0.00% | 2 |
| t8 | recommendation | 0.00% | 1 |

to explore feedback topics, have been rated according to their helpfulness. This number does not allow us to generalize the relation between topics and helpfulness. Nevertheless, to catch a glimpse of possible relationships, we calculated the average helpfulness per topic for this set. As shown in Table 4.8, *feature requests*, *feature information*, and *praise* are the topics included in the most helpful feedback, while *recommendation*, *promise*, and *dissuasion* are included in the least helpful feedback.

Last, we investigated relationships between feedback rating[12] and helpfulness. A $\chi^2$-test showed that these variables are statistically dependent in our data set ($\chi^2$=11952.97, p<0.001). To further analyze their relation, we calculated the average helpfulness per feedback rating. Figure 4.12 shows the result in the form of a sequence of violin plots, one for each rating. Violin plots deliver a similar message as box plots, but also illustrate the probability density of the underlying data. The mean helpfulness is marked with a red line for each rating. By using a series of two-sample t-tests, we could show the following significant relationships for the mean feedback helpfulness according to its rating: $m_1 < m_2 < m_3 < m_4 < m_5$ (p<0.001).

This result means that feedback which rates the application better, is considered more helpful by other users. One interpretation of this result might lie in the main use case of the AppStore, which is to allow users to find good applications. We think that users browse existing reviews in order to understand if an application is perceived to have a high quality by the user community. In this case, their risk of buying a pig in a poke is quite low. Therefore, users might

[12]We excluded the six 0-star ratings from our data set.

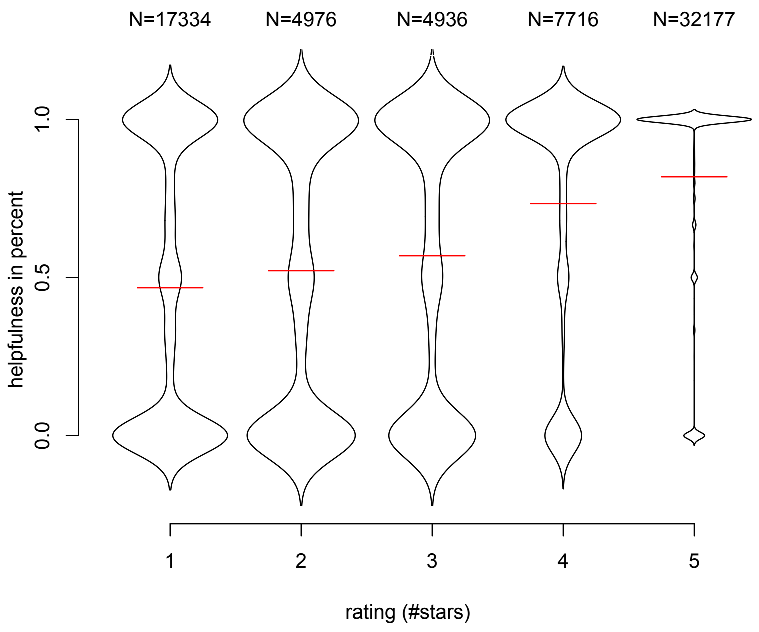


Figure 4.12: Feedback helpfulness and rating. $N = 67139$.

consider better reviews, which typically describe application features and use cases as we found out, as better decision support, and consequently rate them more helpful. On the contrary, it is hard to imagine that users will take the time to rate feedback as helpful which gives the application a low rating. We think that in this case users will rather leave the download area and browse for an alternative.

## 4.5.3 Results Validity

### 4.5.3.1 External validity

Although our study was neither designed to be generalizable nor representative for all application distribution platforms, we think that the results have a high degree of generalizability, in particular for the two remaining big and comparable application distribution platforms Google Play and Microsoft Windows Phone Store. We purposefully decided to study only the Apple AppStore since we had prior experience with the technology and could access the data. At the design time of the study, we preferred a profound hypothesis-generating exploration of one platform over the generalizability to other platforms, in order to provide a starting point for understanding the contained user feedback, its usage, and the impact on market and user community.

Although the entire population is unknown, we think that our results are representative for user feedback the AppStore due to the following reasons.

- Our dataset includes all user feedback from the 1,100 most downloaded applications. It covers feedback from over 4 years.
- We conducted statistical tests to check the statistical significance of our results and exclude hazard factors.
- We got similar results using different analysis methods (e.g. descriptive statistics and content analysis).
- Some of our findings confirm other research results.

Nevertheless, there are three limitations which should be considered when interpreting the results.

First, we obtained our data set by scraping user feedback from the AppStore based on a list of the *most downloaded* applications. This might have affected the resulting data set as relationships between download numbers and other variables, such as in particular the ratings, cannot be excluded. But since our data sample still contains a large number of low ratings (177,887 or 15.79% reviews with less than 3 stars), we are confident with our results. Specifically, our study does not answer questions on absolute numbers within the population such as the overall average rating of feedback. On the contrary, we are interested in how ratings relate to other variables like content and helpfulness, what can be answered with our data sample.

Second, in order to explore topics contained in user feedback, we extracted a smaller sample of 1,100 reviews from our data set, since it was not feasible to perform a manual analysis of the complete data set. Consequently, the statistical evidence is less strong than for the complete data set. We tried to limit this threat in three ways. First, we chose a sample size which allows us to make predictions about our data set at the 95% confidence level, accepting an error margin of 3%. Second, we selected a stratified random sampling strategy, which guarantees that the resulting sample equally considers applications in different categories and with different prices. Third, two researchers analyzed the content independently in order to exclude any hazard factors that would further reduce confidence.

Third, to relate feedback helpfulness and content we could rely only on a very small number of reviews. Although we explicitly avoided any generalization of the obtained results, other results such as relations between ratings and helpfulness indirectly confirm them, what increases our confidence. However, we encourage other researchers to replicate our study and to purposefully select feedback which is rated regarding its helpfulness to perform the content analysis.

#### 4.5.3.2 Construct and Internal Validity

We made the following simplifying assumptions during our analysis, which might partly limit the construct and internal validity of the results:

- Some reviews did not explicitly specify the publishing date, so that we needed to interpolate these values. Since we obtained the feedback in the order it had appeared online, we could compensate for the missing values by investigating the date of the reviews that were published directly before and after. We are confident that this does not influence the resulting data, since feedback date does only include the day and the data gap was never larger than 1 day. In particular, this algorithm never changes the order of feedback.

- To analyze feedback content, we relied on manual analysis. The results are subject to experimenter bias. To reduce this risk, two researchers conducted pair analysis independently from each other. We iterated this activity by refining the rating criteria to improve the inter-raters agreement. We only reported on results where the rates of inter-rater agreement were over 90%.

### 4.5.4 Related Work

Application distribution platforms are a recent phenomenon. Nevertheless, there are already a couple of studies about application distribution platforms.

Chen and Liu [64] present a preliminary study on the popularity of applications in the AppStore. They found that the top ranked paid applications are not necessarily closely correlated with customer ratings, what is confirmed by our study. On the contrary, Harman et al. [126] mined the Blackberry app store for technical, customer, and business aspects of applications. The authors found a strong correlation between an application's ratings and its download numbers, while no correlation seems to be present between price and rating as well as price and download numbers. While we found that application ratings and ranks are not statistically independent in our data set, we were not able to show a correlation. Instead, we described relationships between price and feedback length, and showed that users write more feedback for higher-price applications.

Zhou et al. [296] study several third-party marketplaces for Android to investigate the phenomenon of repackaged applications. The authors found that developers frequently repackage legitimate apps from the original Android store to distribute them on third-party marketplaces. Zhou et al. showed that as many as 5 to 13% of the apps hosted on third-party marketplaces are repackaged and that the main use of repackaging is to replace existing in-app advertisements or embed additional ones to "steal" or re-route advertisement revenues, which illustrates the increasing importance of application distribution platforms as business models. Yamakami [293] profounds the analysis of application distribution platforms from the business point of view. The author presents the major underlying business models and key success factors. Specifically, Yamakami illustrates that viral marketing depends on the fact that users talk about their application *experience* in both the application distribution platforms and their social networks. Our study confirms this finding and additionally shows that the descriptions of user experience are typically missing in negative reviews.

Chandy and Gu [61] aim at classifying spam in the AppStore, in the light of recent bogus reviews. Such reviews can deceive users to download spam apps or to ignore apps which are victims of negative review spam. The authors present a latent class model which is able to classify apps, developers, users, and reviews into normal and malicious categories. Our study confirms that reviews have an impact on the market and the user community. Moreover, it confirms the importance of being in the top lists, since top downloaded apps are not necessarily rated better in the AppStore. However, research is working on putting things right. Researchers like Hong et al. [139] work on the automatic classification of reviews that allows to automatically distinguish helpful and useless reviews.

### 4.5.5 Summary

We observed that users frequently and continuously provide feedback in application distribution platforms. While individual users write only occasionally, the user community as a whole constantly produces dozens of reviews per day and application. Not surprisingly, we found that free applications reach more users and, as a consequence, obtain more feedback than paid ones. But independently from the application price, our results show that most feedback is provided shortly after new releases, with the frequency decreasing quickly over time. One interpretation of this behavior is that users take a new release as occasion for checking if an application meets their expectations, for instance because they were unhappy before. This implies that they expect an appropriate application quality and functionality, and further that they are willing to let developers know if the application fulfills these requirements.

We found that application reviews are generally brief messages – similar to tweets. Interestingly, users tend to write more feedback for higher-price than for lower-price applications, which might indicate that a higher price represents an incentive for users to get more involved. At the same time it could indicate that more expensive applications include more complex features, so that users need to write more to express their concerns. Moreover, feedback length seems to be related to ratings, as we found that users tend to write shorter feedback the more they like an application. One possible interpretation of this result is that less feedback means that there is less to improve, which would imply that information contained in user feedback mainly addresses shortcomings.

In spite of its briefness, user feedback typically contains multiple topics about software maintenance, the user community, application ratings, and user experience. An important finding which confirms our results described in Chapter 3 is that users often communicate maintenance relevant shortcomings to developers and request improvements and additional features. In addition, most feedback content includes ratings in natural language, but users also frequently share their experience with the application. We found that these two themes occur particularly often together, which indicates that users tend to give reasons for their ratings. One implication of this finding is that users are aware of the rating potential and willing to provide helpful explanations to the application developers. Indeed, our study confirms that feedback content has a real impact on the market, since more positive messages usually also lead to better application ratings and vice versa. Given that in particular bug reports and shortcomings are market critical feedback, it is unfortunate that user experience is often missing in such negative reviews, because it reduces the likeliness that developers will be able to improve their applications from this feedback alone [298].

We found that feedback quality varies widely, from helpful messages for other users and developers to insulting offenses, for instance when formerly beloved features were removed in a new release. In general, positive feedback is considered more helpful by the user community. This can be explained by the main use case of application distribution platforms which is to support users in finding good applications. Likewise, longer feedback is more likely to be regarded helpful by other users, probably because the possibility to contain helpful information is generally higher in longer text. An important finding is that the community typically agrees about the helpfulness of the provided feedback. This justifies the hypothesis that user communities are capable of aggregating helpful feedback.

## 4.6 Summary

The studies we have described in this chapter show that users are willing and able to provide helpful feedback to developers when they are convinced that it improves the software they use. But we also identified serious shortcomings in the current feedback channels which prevent the benefits of a more thorough user-developer communication.

- In open source communities boundaries between users and developers are low – if they exist at all. Social media which allow both users and developers equally to express their opinions, seem to help dissolving such boundaries, as we did not discover any *major* and *significant* difference between developers' and other stakeholders' blogging habits. Both groups *frequently* rely on social media to publish project-related information on a *similar level of abstraction* and formality. Also, both discuss about *similar topics* such as *requirements*, implementation, and community aspects. Developers on the one hand report about their *recent development activities* to communicate their project work to a broad audience, including the users. On the other hand, users and other stakeholders have their blogging peak time shortly after new versions are released and report about their *experiences* with the software.

- Commercially used user feedback channels, such as application distribution platforms only allow for *transactional* communication which prevents otherwise emerging benefits. Similar as in open source communities, users *frequently* provide feedback in particular shortly after new releases, but their feedback is much *shorter* than blog posts and its *quality* tends to be lower. Users often include maintenance relevant *shortcomings* in their feedback and *request improvements* and additional *features*. They also share their *experiences* with the application, but typically only in positive feedback, what reduces the likeliness that developers will be able to improve applications only from such feedback.

Based on these results, we include the following four major factors for successful continuous user involvement during software evolution in our grounded theory:

1. Users should be *personally* involved, i.e. their feedback should represent their *personal* opinion, as it does in social media. This could prevent unqualified content, allows developers to contact a specific user, and enables other users to take part and comment.

2. Communication between users and developers should be *bidirectional* as in open source communities. This allows developers to directly react to users, which is not possible currently. Developers need to be able to provide users with clarifications and to ask for clarifications and missing information themselves.

3. User *communities* should be fostered systematically with the goal to obtain a measure for feedback impact. The emerging synergies between the opinions of different users should lead to more perspectives, focused, and mature input.

4. User *experience* should be captured automatically, since users tend to omit this from negative feedback – where it is most needed. Specifically, feedback is most useful if the use *context* and the use *history* are known to the developers.

In Chapter 5 we will introduce a domain-independent model for continuous user involvement which satisfies these properties. Since we found that users do not tend to include their experience with an application in negative feedback, we follow a proactive approach. We also describe in detail how this approach allows us to consolidate user feedback by exploiting recurring information entities as well as the application use context.

# Chapter 5

# Proactive and Context-Aware Recommendation of User Feedback

*«If you have an apple and I have an apple and we exchange these apples then you and I will still each have one apple. But if you have an idea and I have an idea and we exchange these ideas, then each of us will have two ideas.»*

— George Bernard Shaw

In Chapter 3 we found that post-deployment user feedback is helpful and important for developers, but also that its analysis requires high effort. In order to assess the impact of user feedback, developers need to consolidate and group it according to the user experience described, which is often insufficient or even missing. In Chapter 4 we found that user communities can help with such an assessment and concluded that user experience should be gathered automatically since it facilitates the consolidation.

In this chapter, we describe a solution for the consolidation and prioritization of user feedback which harnesses the *user community* to group user feedback in a *proactive* way, while increasing its overall quality. To avoid the creation of unstructured, duplicate feedback, we recommend existing *relevant* feedback to users, which they can rate, vote for, or comment on instead of creating feedback themselves. The relevance of existing feedback is estimated by comparing users' experience with an application based on the corresponding observed use contexts. We call this approach *proactive and context-aware recommendation of user feedback*, or Portneuf.

In Section 5.1, we present the domain-independent Portneuf model, and introduce its main abstractions (user experience, collective user feedback, feedback recommendation, and feedback impact) as well as their structural and dynamic relationships. In Section 5.2 we describe three different applications of Port-

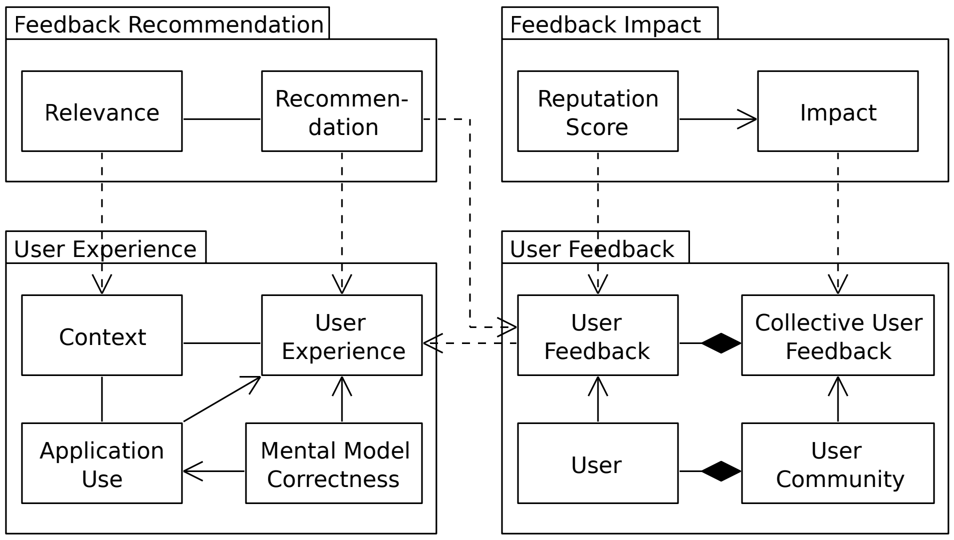


Figure 5.1: Overview of the Portneuf user involvement model.

neuf in common software engineering activities. Section 5.3 introduces the Portneuf framework architecture and explains recommendation and impact assessment algorithms. Section 5.4 discusses related work, focussing on user feedback research and existing user feedback systems. Section 5.5 summarizes the chapter, revisiting the most important properties of Portneuf.

## 5.1 Portneuf Model

Figure 5.1 shows an overview of the domain-independent Portneuf model, which we will describe in this section. In order to realize the four key factors for continuous user involvement, the Portneuf model introduces *user experience* as main abstraction (Section 5.1.1). With Portneuf, individual *users* describe their experience in *user feedback*, while the *user community* as a whole provides *collective user feedback* (Section 5.1.2). Portneuf allows to group similar feedback and avoid duplicates by creating feedback *recommendations* based on its *relevance* in a specific context (Section 5.1.3). Finally, the model enables developers to assess the *impact* of specific feedback based on the opinion in the user community and the individual *reputation* of the reporters (Section 5.1.4).

### 5.1.1 Model of User Experience

Users' experience with an application depends on multiple factors, as illustrated in Figure 5.2. *Using* an *application* in a specific *context* influences users' perception of their *mental model correctness*. This perception together with the

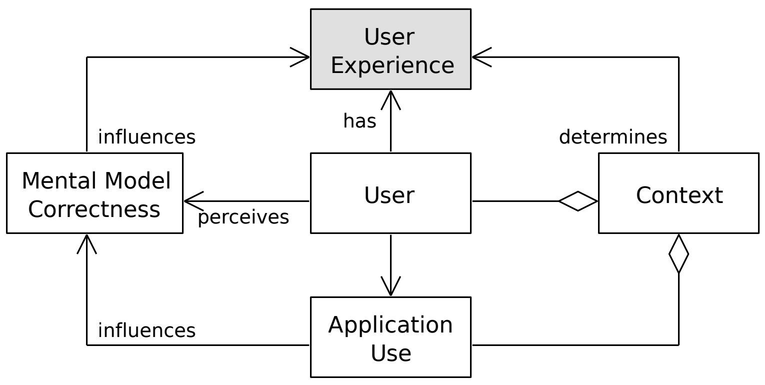


Figure 5.2: Main abstractions of user experience.

particular context of use determine *user experience*. Consequently, in the following we model application use, context, and mental model correctness before introducing our model of user experience. We derive these models mainly from research insights in human-computer interaction, usability engineering, and context aware software engineering.

#### 5.1.1.1 Application Use

We model application use as composition of both *user interaction* and *application execution*, as illustrated in Figure 5.3. The obvious reason for this combination lies in the fact that both user and application are involved in the use of an application on an interactive computer system. Typically, application execution brings forth a result (e.g. on a user interface) on which the user can react with an interaction (e.g. by clicking a button), thereby causing further application executions. This dynamic behavior has been termed a "dialogue" in human-computer interaction [55]. With this model we also capture the two major information needs in software maintenance [298]: To understand how an application was used, developers need information about user interactions (e.g. steps to reproduce an error) and about the application execution (e.g. stack traces). Application use consists of the ordered sequence of user interactions and application executions, which is why it can be seen as use "history". We chose this retrospective perspective because the model needs to capture events that already have happened.

**User Interaction** Human-computer interaction always happens at the *user interface* [140], which includes both hardware and software interfaces. For the purpose of this thesis, only software user interfaces are considered. User interaction refers to the interaction of a user with an application via the user interface and denotes a two-way causal effect, as opposed to a single-way action [179]. Users typically interact multiple times with an application while using it. For

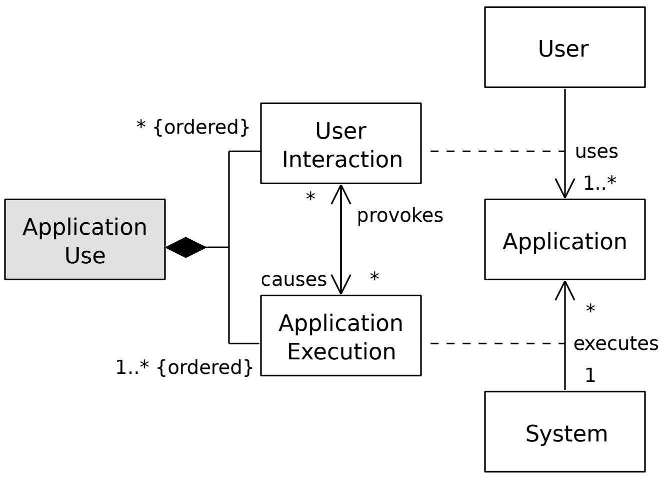


Figure 5.3: Model of application use.

example, they might start a word processor to edit a document, select text in the document, click on a button to increase the font size, and so forth.

**Application Execution** With application execution, we denote the working activity of an application. Application execution is a time-constrained, abstract, multi-granular concept, which can be broken down from the conceptual notion up to a single execution step of the processor. Software engineers define how an application executes by modeling its *behavior* with dynamic and static models, and by implementing this behavior in the form of *classes* and *methods* [49]. Maintenance engineers investigate application execution retrospectively e.g. by investigating method calls in *stack traces* [48, 298]. A large research fraction is concerned with enabling and improving the deterministic replication of specific application executions [80, 185, 204, 225, 267].

How a user uses an application depends also on her current *context*, which is particularly evident in the case of mobile applications. Consequently, we introduce context as major abstraction in the PORTNEUF model. Figure 5.4 shows how context influences application use in the model, and which concepts it includes.

**Context** According to Merriam-Webster, context describes “the interrelated conditions in which something exists or occurs” [195]. In computer science, the term mainly refers to conditions under which a system operates, that are relevant to the system behavior. Definitions have focused on different types of conditions, at first mainly on physical, environmental, and user-specific circumstances. Schilit et al. [256] coined the term *context-awareness* in the ubiquitous computing area in 1994. The authors defined context in particular as location of

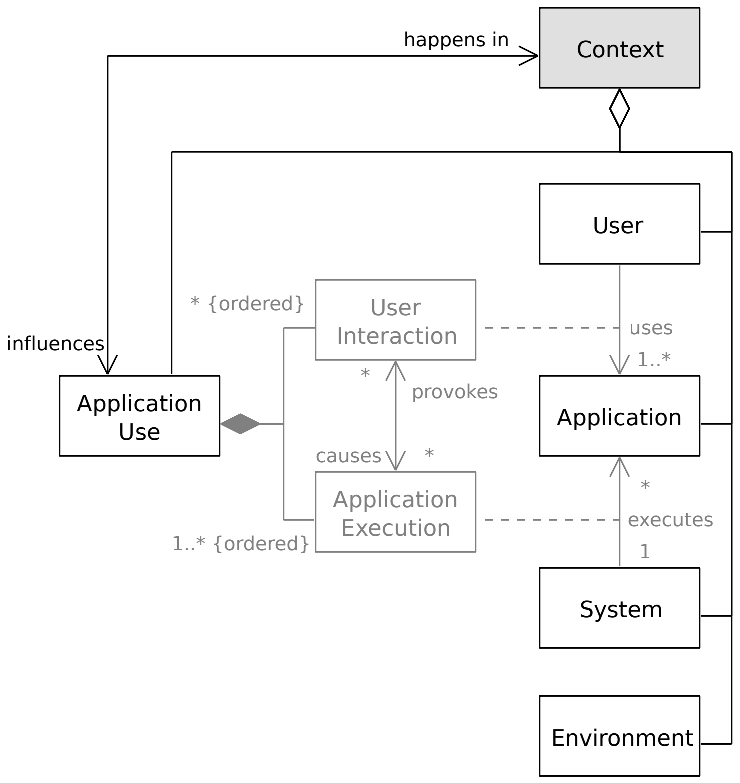


Figure 5.4: Context-aware model of application use.

use, nearby objects (e.g. people), and changes to these entities over time. In the following research the definition shifted to system users, their physical but also emotional state, and their environment [47, 81, 257]. Dey and Abowd's definition of context [82] is information-centric and focuses more on the interaction of users with an application. The authors define context as any information used to describe a person, place, or object relevant to the interaction between a user and an application, including the user and applications themselves. Maalej defines context in a work-centric way. He summarizes the context in which software engineering work is performed as "the set of all events and information, which can be observed and/or interpreted in the course of the work, except those events and pieces of information that constitute the change (i.e. the main output of the work)." [179]. Our definition of context does not aim for a complete description of its elements, but rather frames its necessary properties. It takes into account any influence on application use and is particularly motivated (a) by the fact that both user interaction and application execution happen in a specific context and (b) by mobile applications which are used under changing circumstances.

**Definition 3. Context** refers to all current and past conditions and events which *influence* the interaction of a user with an application or the execution of an application on a system.

When building context-aware systems, one property of context is indispensable: observability [179]. At any point in time, any user interaction and any application execution happens in a specific context. However, only observable contextual information is useful for context-aware systems, because it can be exploited to change the application behavior.

#### 5.1.1.2 Mental Model Correctness

According to Norman [208], users develop a mental model of an application while using it. System designers try to anticipate the users' mental model and realize application user interfaces and workflows based on this prospect, called "design model" by Norman. Thus, the design model depends on a hypothetical mental model created by designers, not end users. Problems occur if designers make wrong assumptions, i.e. when the hypothetical mental user model does not match with the real one.

As shown in Figure 5.5, users form their mental models by *observing application behavior*, i.e. by interpreting the visible application structure as well as the perceived application execution [208]. At the same time, users compare the observed application behavior to the *expected application behavior*, i.e. how they think the application should look like and how it should react on specific interactions. From a user perspective, the difference between expected and observed application behavior denotes if their mental model is "correct", in the

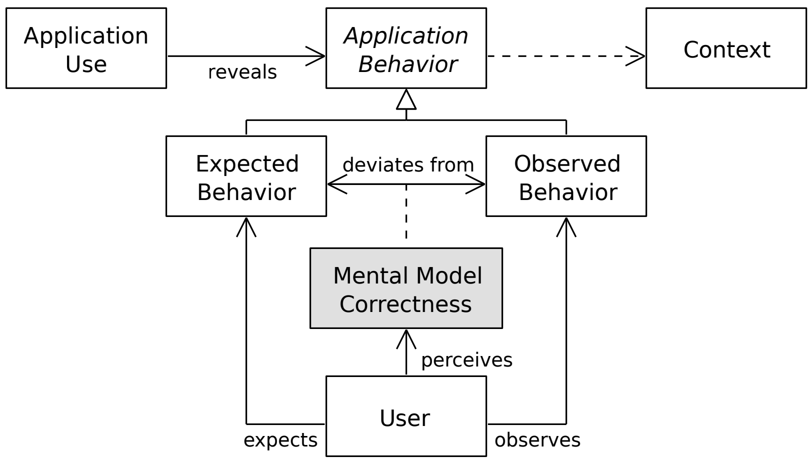


Figure 5.5: Model of expected and observed application behavior.

sense that it corresponds to the designers' assumption. Users obviously perceive how correct their mental model is: If it fits well, the application will result user-friendly and highly usable, while otherwise it will be considered unusable. Consequently, the users' mental model correctness influences their experience. Besides, expected behavior changes over time while the user's mental model evolves. Therefore, new users have different expectations than experienced ones. Recently researchers such as Paternò et al. [228] and Vargas et al. [279] exploit these facts in order to identify usability problems by analyzing user behavior.

Note that our definition of mental model correctness is *user-centric*, i.e. it refers to correctness from users' point of view. This perspective is appropriate for *software evolution*, where the system is already deployed. It might not be for traditional *software design*, where mental model correctness is seen from the designer's point of view, and refers to the correctness of the assumptions made.

#### 5.1.1.3 User Experience

In computer science, the term user experience was first coined in 1995 by Norman et al. [210] in an organization overview of Apple's Advanced Technology Group (ATG), where the authors used it to express their approach to human interface research and application. Until today there is no commonly accepted definition of user experience. There are even movements to collect the various user experience definitions [248] and to come up with a definition inspired by the community [247]. According to Hassenzahl and Tractinsky [132], the term is associated with a variety of meanings, "ranging from traditional usability to beauty, hedonic, affective or experiential aspects of technology use". But unlike usability, which relates to more pragmatic concepts like effectiveness and efficiency of use as well as subjective user satisfaction [143], most researchers consider user experience to be much broader in scope.

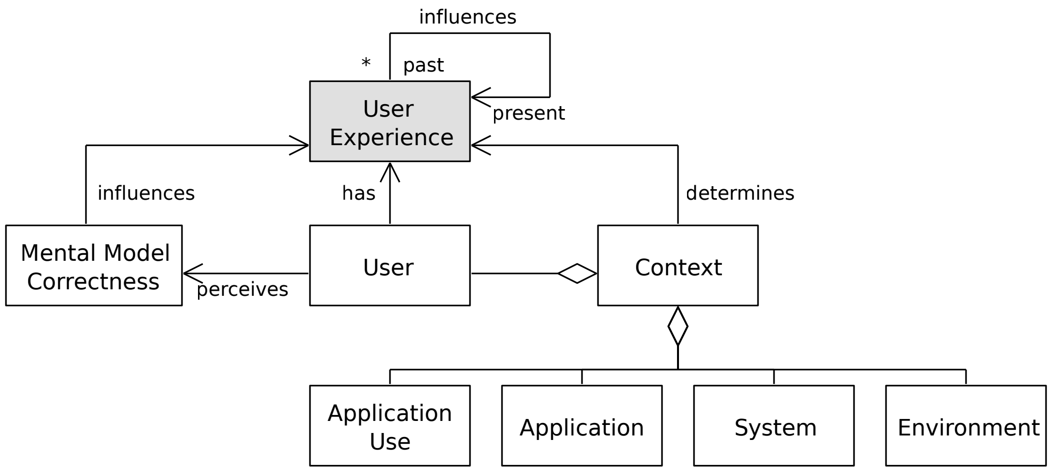


Figure 5.6: Context-sensitive model of user experience.

According to an early definition by Alben [6], user experience covers "all the aspects of how people use an interactive product: the way it feels in their hands, how well they understand how it works, how they feel about it while they're using it, how well it serves their purposes, and how well it fits into the entire context in which they are using it". Several researchers have further published their views of user experience, focusing on different aspects like enjoyment [131] and social interactions [20], or specific platforms such as the web [108] and mobile devices [60] (for an overview refer to [103]). According to Hassenzahl [130], experiences are unique to individuals and change over time. More specifically, Kankainen underlines the importance of previous experiences, which may influence the present user experience [155]. Consequently, our model includes past and present user experiences, as shown in Figure 5.6.

Law et al. [170] conducted a scientific study among researchers and found that most respondents agreed that user experience is dynamic, *context-dependent*, and subjective. The ISO standard 9241-210 [145] defines user experience as "a person's perceptions and responses that result from the use or anticipated use of a product, system or service". The first authors to collect influential components on user experience were Forlizzi and Ford [104]. According to them, user experience is influenced by the user, the product, the context of use, and social and cultural factors. The first three factors can also be found in other definitions of user experience [145, 170, 247] as well as in definitions of usability [143]. Consequently, we model user, application, and other context as important factors for user experience.

**Definition 4. User experience** describes a user's perception of a system at a given point in time and in a specific context. It is mainly determined by this context and in particular by the application use. It depends on the user's past experience and mental model.

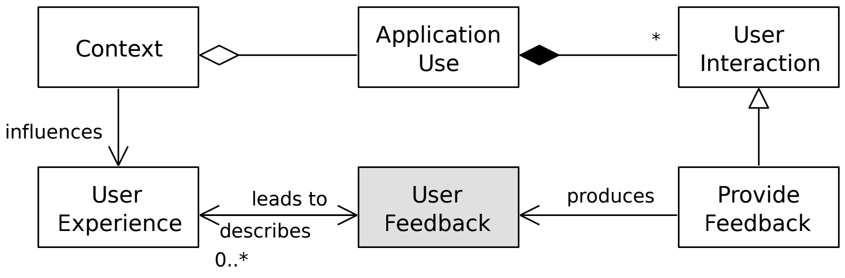


Figure 5.7: Model of user feedback and user experience.

Since user experience is a sensation subjective to a human being, it is not directly measurable by means of an algorithm with current technology. Nevertheless, the main influential factors, specifically the *context* and in particular *application use*, can be observed. This allows an *indirect* measurement of user experience. Capturing user experience is a prerequisite for the functionality of Portneuf. Consequently, we formalize our assumption about the relation between user experience and context in the following hypothesis.

**Hypothesis 5.1.** *Let* $ux(u,a,c,t) : U \times A \times C \times \mathbb{R} \rightarrow \mathbb{UX}$ *be the user experience of user* $u$ *with application* $a$ *in context* $c$ *during the time interval* $t$*. Let further* $d_{UX} : \mathbb{UX} \times \mathbb{UX} \rightarrow \mathbb{R}$ *be a metric on user experience. Then, there exist* $n \in \mathbb{N}^+$*, a computable function* $\gamma : U \times A \times C \times \mathbb{R} \rightarrow \mathbb{R}^n$*, and a homomorphism* $\Upsilon : \mathbb{R}^n \rightarrow \mathbb{UX}$*, such that* $d_n(\gamma_1, \gamma_2) = d_{UX}(\Upsilon(\gamma_1), \Upsilon(\gamma_2))$ *for two observed contexts* $\gamma_1, \gamma_2 \in \mathbb{R}^n$ *with* $\gamma_1 = \gamma(u_1, a_1, c_1, t_1)$ *and* $\gamma_2 = \gamma(u_2, a_2, c_2, t_2)$*.* □

Hypothesis 5.1 claims that comparing two observed contexts of use in an appropriate way suffices to compare the underlying user experience.

### 5.1.2 Model of User Feedback

As shown in Figure 5.7, we model the activity of providing feedback as particular type of user interaction. This is always the case for in-situ feedback which is given while actually using the application itself. Enabling in-situ user feedback is one particular goal of the Portneuf model. We distinguish between the activity of providing feedback, and the actual user feedback which is produced during this activity.

**Definition 5.** Providing feedback is a user interaction carried out by the user to communicate her subjective experience with the application to the application developers with the goal to improve the application. The resulting, user-generated artifact of this activity is called **user feedback**.

Whether a specific user feedback eventually contributes to improve the application is not relevant in our definition, since it is not clear at the time of feedback provision.

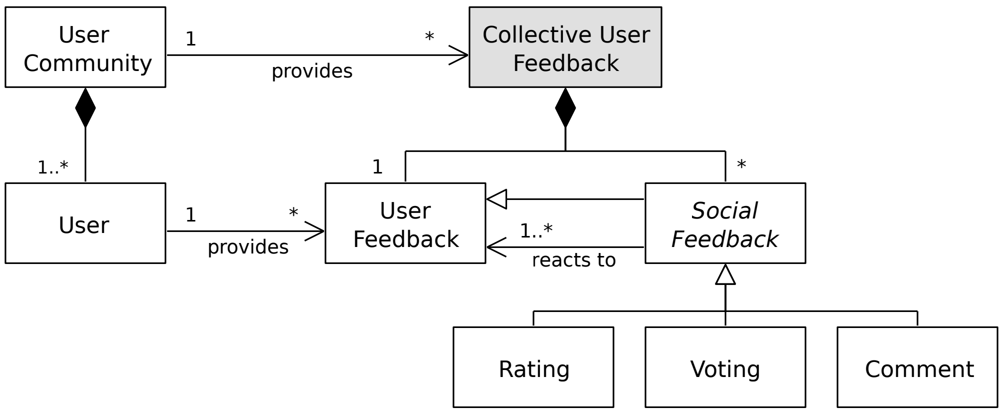


Figure 5.8: Model of collective user feedback.

It is important to note that *what* users report depends on their prior experience with the application. For instance, we assume a user will only file a bug report if she has experienced a bug, or request a feature after missing it while she was using the application. Recent empirical studies suggest that user feedback describing prior user experience is particularly useful for designers and developers [216, 244]. Reasons for this include that user feedback describing prior experience is typically concrete and detailed, often provides "clear examples of specific, contextual issues which are hard to imagine", and contains personal stories which are felt trustworthy. In particular concrete information about the application use and the context of use are considered useful. In order to fix errors, for example, concrete descriptions of user experience such as the steps that lead to the error, as well as expected and observed behavior are of particular help for developers [298]. Unfortunately, research has shown that user feedback which contains this information is the exception rather than the rule [298]. In Chapter 4 we found that a possible reason for its absence lies in users' disappointment when they have to report shortcomings. But even if users include their experience in their feedback, it is often not correct or not sufficient for developers to reproduce the reported situation.

#### 5.1.2.1 Collective User Feedback

As shown in Figure 5.8, *collective user feedback* consists of feedback from multiple users within a *user community*. First, a specific user provides user feedback describing her experience. Second, other users provide *social feedback*, typically as a reaction to already existing user feedback or other social feedback.

**Social Feedback** We use the term social feedback to indicate feedback which is given as reaction to other feedback. The term is inspired by the lightweight feedback mechanisms typically used in Web 2.0 communities. These include

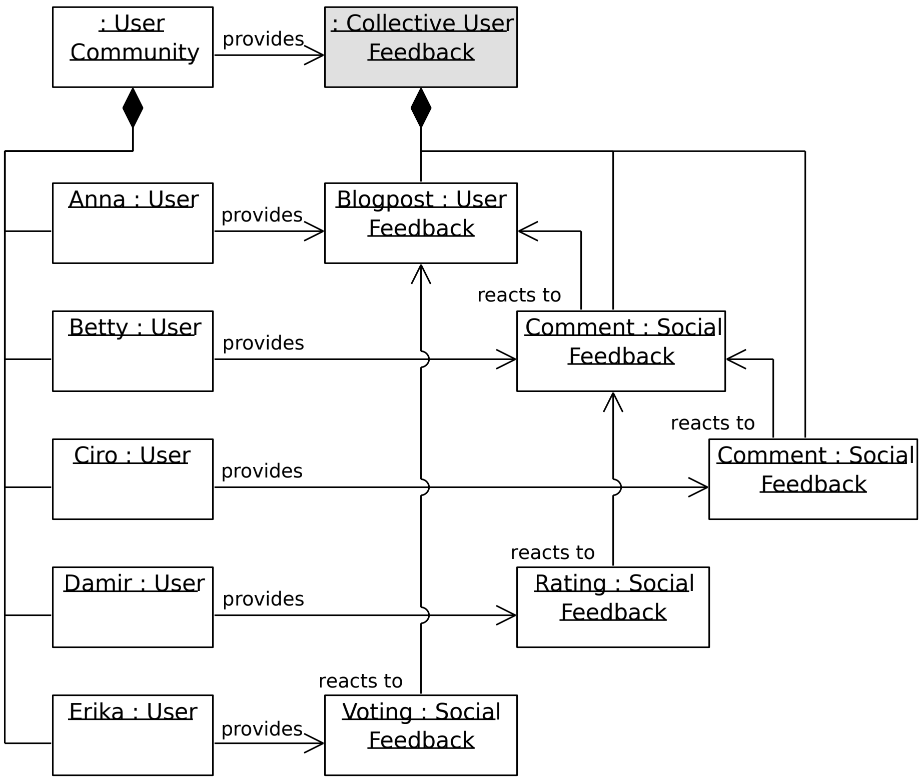


Figure 5.9: Example of collective user feedback.

ratings, votings, and comments [176]. *Rating* refers to the assignment of an ordinal value, and can be expressed for instance using a Likert scale [174] or an increasing number of stars. *Voting* denotes assigning a nominal assessment, and can be expressed for example using a like button [234]. *Comments* consist of text that typically refers to existing prior user feedback or parts of it, where the author expresses her opinion using natural language. In general, social feedback reacts to already existing feedback and is utilized to express the author's positive or negative opinion about it. Since rating and voting allows users to express their opinion about a specific prior user feedback in a *quantitative* way, collective user feedback in the best case enables the aggregation and thus quantification of the user community's opinion about a specific topic. In other words, it allows developers to assess the *community impact* of specific feedback

**Definition 6. Collective user feedback** consists of a specific user feedback, written in natural language, and the transitive closure of all social feedback which emerged as a reaction to this user feedback.

Figure 5.9 gives a concrete example of collective user feedback as it can be found in blogs. User Anna published a blog post where she describes her user experience. Next, user Betty commented on Anna's blog post. After that, user

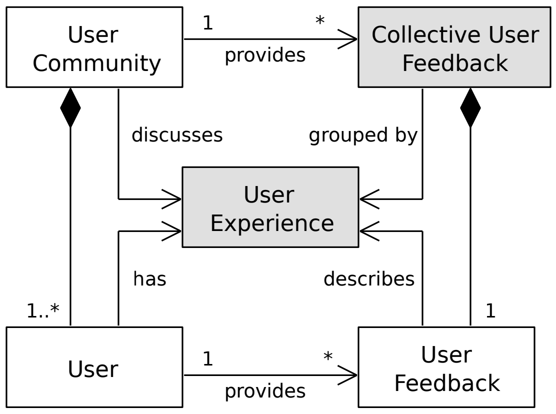


Figure 5.10: Collective user feedback and user experience.

Ciro provides a comment to Betty's previous comment, while user Damir rates it. Last, user Erika votes for the initial blog post.

#### 5.1.2.2 Collective User Feedback and User Experience

We model user experience as the central grouping concept for collective user feedback, as shown in Figure 5.10. As we have seen above, user feedback describes prior user experience. With the additional social feedback other users express their opinion about this user experience and about the conclusions drawn by the previous authors. This makes user experience the central "topic" discussed by the users contributing to collective user feedback, and gives rise to a common, collective user experience. While "co-experience" [20] is user experience which is *created by* social interaction, collective user feedback refers to different individual experiences which are grouped around the same application.

Combined with user experience, collective user feedback can be seen from three perspectives. It represents *communication* between users and developers, contains *issues* experienced by users, and allows for user *innovation*.

**Communication** Collective user feedback can exhibit a communication structure similar to a *discussion*. Typically, a user called the *reporter* first publishes user feedback which is read by other users afterwards, who then react to the feedback by providing social feedback for their part. Consequently, collective user feedback exhibits the same structure as a discussion *thread* about a specific user experience *topic*. In addition to traditional discussions, social feedback allows users to express and quantify agreement and disagreement.

**Issues** In rationale management, an *issue* represents a concrete problem to be solved [49]. Additionally, issues carry information about possible solutions (*alternatives*), desirable qualities to be satisfied by a valid solution (*criteria*),

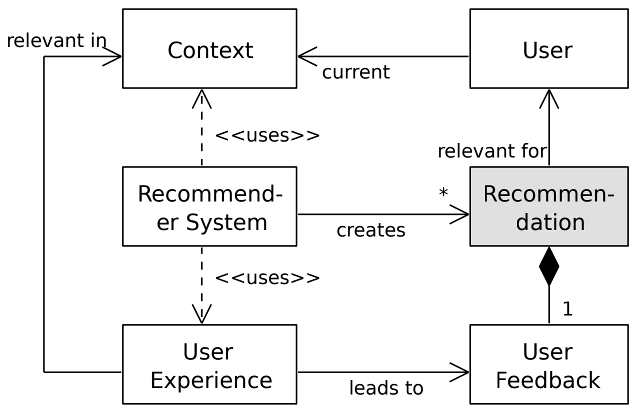


Figure 5.11: Context-aware model of user feedback recommendation based on user experience.

*argumentations* for different alternatives by different stakeholders, and eventually a final *decision*. Collective user feedback can describe similar information. The initial user feedback typically describes a concrete problem of the user based on her experience with the application. Often, also argumentations for an alternative are given, and in some cases even justified by arguing with criteria. Therefore, collective user feedback can be seen as user generated issue with multiple argumentations by different stakeholders for different alternatives. In fact, developers typically create issues from gathered user feedback as we have seen in Chapter 3.

**Innovation** Researchers like Tuomi and von Hippel [275, 283] have shown that many users modify products for their own use in a socially and socio-technically distributed way. Thereby, users particularly strive for innovations that deliver novel functions [240]. Collective user feedback allows users to discuss their experience, express different points of view, and vote for and against specific innovations. This collaboration fosters synergies between the opinions of different users, leading to more perspectives, focused, and mature feedback [182].

### 5.1.3 Model of User Feedback Recommendation

The main goal of PORTNEUF is to group similar user feedback and avoid duplicates by recommending users to provide social feedback on existing feedback. Figure 5.11 shows the PORTNEUF model of user feedback recommendation. A framework implementing the PORTNEUF model creates recommendations of existing user feedback based on the user experience that led to its publication and based on the relevance of this experience within the current context of application use. The following hypothesis underlies the recommendation of user feedback in the PORTNEUF model.

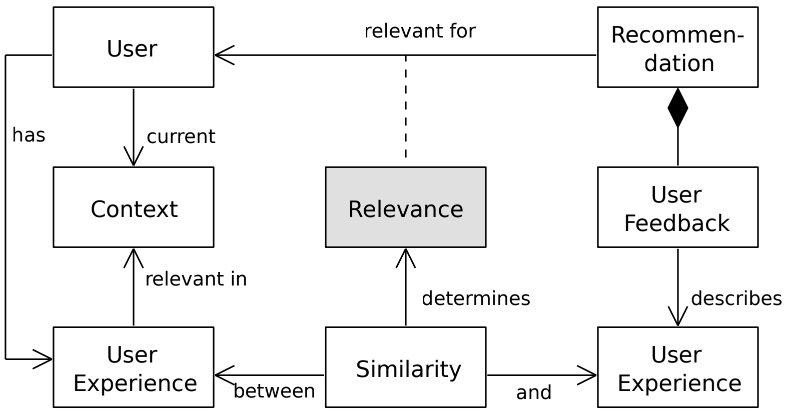


Figure 5.12: Context-aware model of feedback recommendation relevance.

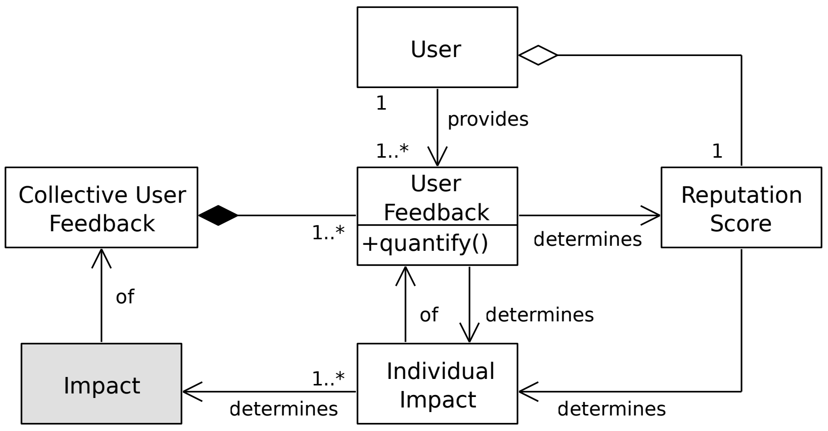


Figure 5.13: Model of user reputation score.

**Hypothesis 5.2.** *The relevance $rel(f, u)$ of the recommended user feedback $f$ for a user $u$ depends upon the user experience $ux(u, a, c, t)$. The more similar the user experience is to the one described in the user feedback, $ux(f)$, the more relevant this feedback is for the user: $rel(f, u) \sim d_{UX}(ux(u, a, c, t), ux(f))$* $\square$

Figure 5.12 illustrates this relation between the relevance of a user feedback recommendation and the similarity of user experience. Note that by Hypothesis 5.1 the relevance of a user feedback recommendation can be measured by comparing the corresponding observed contexts.

## 5.1.4 Model of User Feedback Impact

Another major aim of PORTNEUF is to allow developers to assess the *impact* of specific feedback. As shown in Figure 5.13, the impact of collective user feedback is determined by the individual impact of the contained user feedback, which in

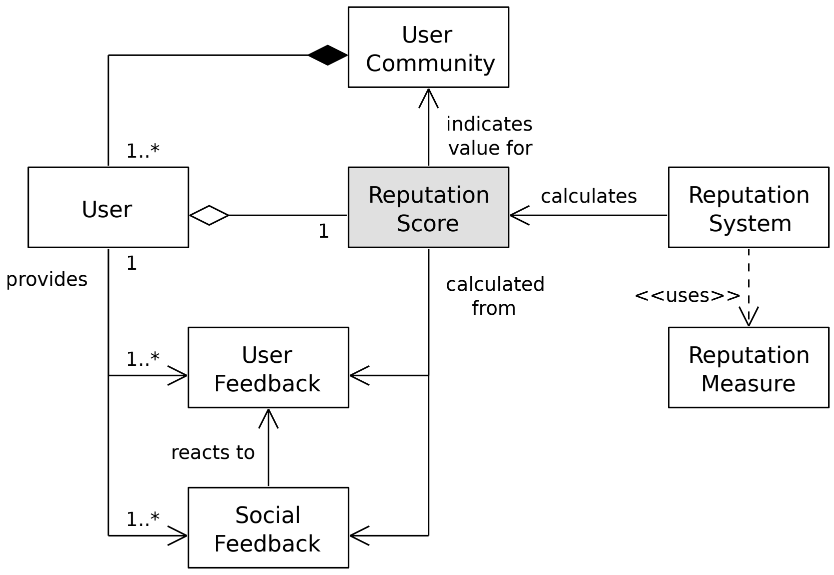


Figure 5.14: Model of user reputation score.

turn depends on two factors. First, the user feedback itself, which is quantifiable either by simply counting it or by considering its value in case of a voting or rating. Second, the reputation score of the user who provided the feedback.

We explicitly model the *reputation score* of a user which corresponds to the user's potential for the user community, for two reasons. First, to provide a measure of trust for developers (cf. Chapter 3), and second, to be able to increase the quality of user feedback. It has been shown that both can be reached by employing reputation systems [154]. Online community users often need to interact with other users whom they do not know in person and with whom they do not have prior experience. In this situation users rely on an estimation of the trust of other users, either based on frequency or based on transitive trust. Moreover, reputation systems are utilized to create incentives for specific desired user behavior, in particular in systems with user generated content. Reputation points are for instance employed to increase the number of contributions, but also to improve the content quality.

As shown in Figure 5.14, we model the reputation score of a user within a community depending on the feedback the user provided (e.g. amount of feedback), and depending on the social feedback which this specific user got from other users in the community (e.g. rating of or voting for her feedback).

This model is inspired by the current Web 2.0 reputation systems, where users can earn *reputation points*[1] and so-called *badges*[2]. Prominent examples include Stack Overflow [1], foursquare [106], or eBay [89]. Jøsang et al. consider

[1]see for instance http://superuser.com/faq#reputation
[2]see for instance http://superuser.com/badges

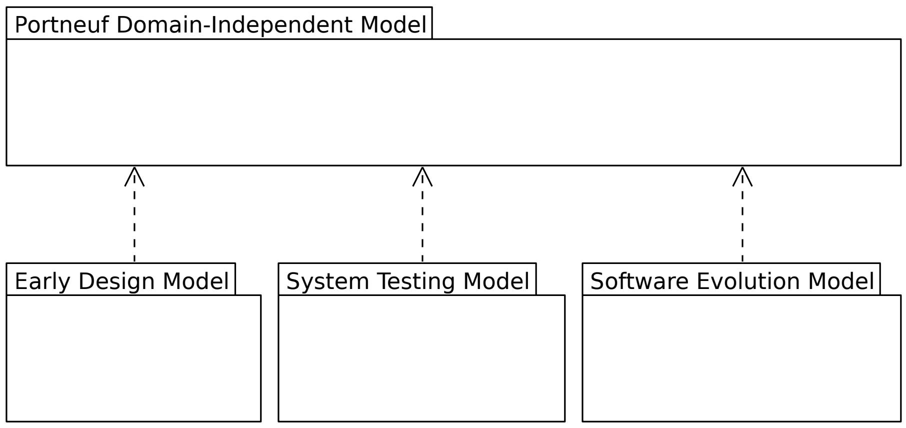


Figure 5.15: PORTNEUF applications.

reputation as "a collective measure of trustworthiness (in the sense of reliability) based on the referrals or ratings from members in a community" [154]. The concrete meaning of reputation and the specific measure applied to calculate it can vary and depend on the situation at hand.

Some of the various measures for computing reputation and trust that have been proposed are employed in commercial applications such as eBay and Google. Basic principles include: summation or average of ratings [238, 258], Bayesian systems [153, 196, 286], discrete trust models [2, 54], belief models [151, 152], fuzzy models [190, 252], and flow models [172, 223, 297]. For a comprehensive description of these measures, we refer the reader to [154].

## 5.2 Portneuf Applications

The PORTNEUF model introduces the abstract concepts necessary to capture user experience and collect user feedback, to group user feedback around user experience into collective user feedback, to generate feedback recommendations based on user experience and context, and to assess the impact of feedback based on its value for the community.

The abstract concepts defined in PORTNEUF are generic and applicable for different situations. More specifically, the model can be instantiated and extended for any application which involves retrospective feedback, i.e. any activity carried out when a system or prototype is available. In the remainder of this section, we present three usage scenarios of PORTNEUF (illustrated in Figure 5.15): *early design*, *system testing*, and *software evolution*, which is the main focus of this dissertation.

### 5.2.1 Early Design

At first glance, early software design does not seem to allow retrospective feedback, since the system to be designed is not yet implemented. However, recently iterative process models and rapid prototyping techniques are becoming mainstream. Developers use prototypes especially in the early design phases to be able to evaluate specific system parts before the system as a whole has been completed [268]. Prototypes are particularly helpful to deal with the IKIWISI ("I'll know it when I see it") phenomenon [39], which states that users are frequently not able to express their needs and expectations from scratch, but quite good at criticizing an existing system. Consequently, prototyping is gaining more and more attention in research and practice. Recent research by Stangl and Creighton even suggests a "continuous demonstration" of the system [269], in order to benefit from feedback as early as possible.

Developers typically evaluate prototypes with potential users, in order to identify shortcomings such as, for instance, usability issues. As discussed in Chapter 2, the employed user involvement methods usually require developers to select a set of users (e.g. focus groups [206]). The size and representativeness of this set decides about the success of the evaluation. It is therefore no wonder that research tries to develop methods to involve as many persons in evaluations as possible, for example by utilizing tools such as the Amazon Mechanical Turk[3] [160]. But with a growing number of users the problems are also increasing.

PORTNEUF provides support for two major problems of large-scale prototype evaluations. First, it enables a structured collection of the emerging issues. PORTNEUF captures the experience of the user with the prototype and provides recommendations for existing similar issues whenever users try to report. Second, it allows developers to quantify the evaluation result by calculating the impact of reported issues among all users.

Figure 5.16 shows how the PORTNEUF model can be used during early system design. In this example, `prototype testers` might judge a prototype to be `confusing` during a usability test. Consequently, `usability test issues` are created that correspond to collective user feedback and contain `usability test alternatives` or `argumentations`. In the following we describe a scenario that shows how PORTNEUF supports early design.

`Prototype tester Laura` performs a `usability test` of the `Account` view in a banking software. Her task is to change the account holder name. She `opens` the `Account` view, which is then `shown` by the system. She `searches` for the account holder field, but cannot find it. After two minutes, the prototype runs into a security `timeout` and closes the form. `Laura`'s impression is that the `Account` view is `confusing`. Consequently, she raises an `issue` to the designers, stating that the `Account` view should be `improved` which is considered

[3]https://www.mturk.com/mturk/welcome

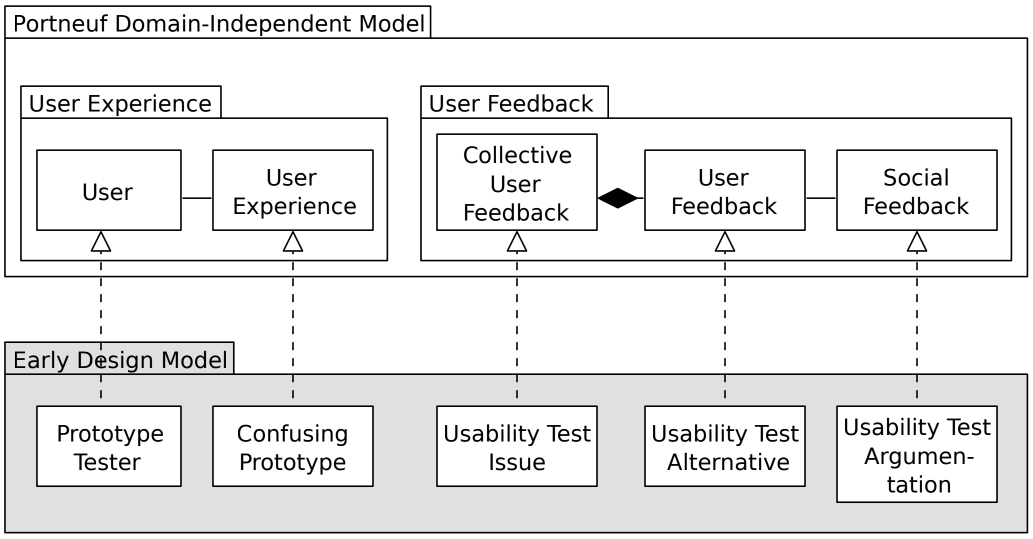


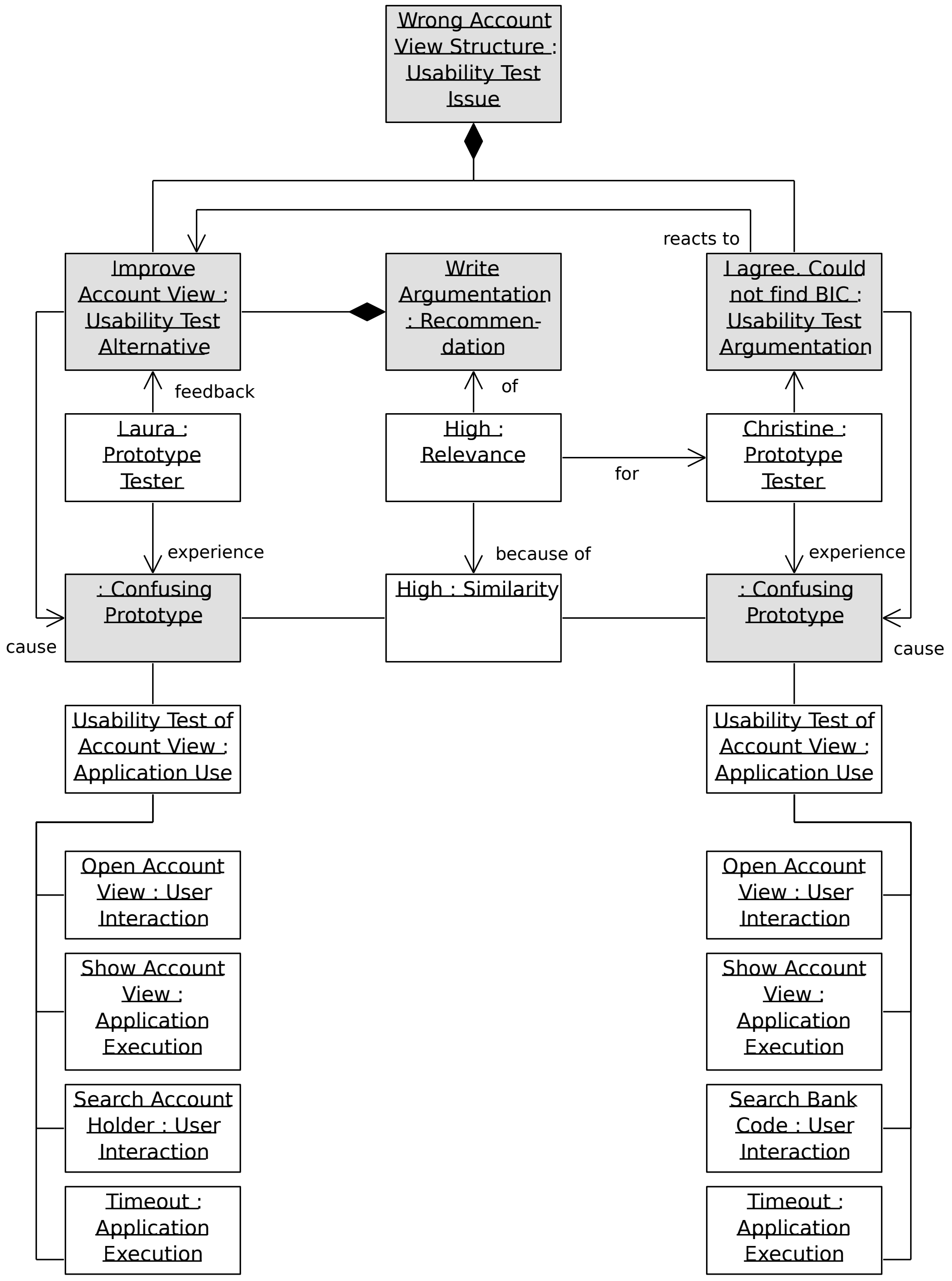
Figure 5.16: Using PORTNEUF during early design.

an `alternative` in rationale management. Two days later, `prototype tester Christine` performs a similar `usability test` for the `Account` view. Her task is to copy the bank code into the clipboard. She `opens` the `Account` view, which is `shown` in the user interface by the system. She `searches` for the `bank code` field on the view, but has problems finding it. After two minutes, the prototype runs into a security `timeout` and closes the form. Similar to `Laura`, `Christine` is not satisfied with the `Account` view as she finds it unintuitive. She plans to raise a corresponding issue, but instead obtains a `recommendation` to write an `argumentation` for the `alternative` reported by `Laura`, who had a similar experience with the `Account` view. `Christine` reads `Laura`'s `alternative` and decides to support it by the `argumentation` that she could not even find the bank code.

Figure 5.17 depicts the resulting objects. The left side of the figure shows `Laura`'s `user experience` and `user feedback`, while the right side illustrates these objects for the case of `Christine`. Note that in this case, two different usability tests led to the same issue, what typically complicates a reactive consolidation of such feedback.

### 5.2.2 System Testing

System tests aim at ensuring that the system under development complies with the specified functional and nonfunctional requirements [49]. Traditionally, developers prepare these system tests with *test case specifications*, which contain inputs, drivers, stubs, together with the expected outputs as well as the concrete tasks to be performed by testers [49]. Tests are executed by *testers*, a



Wrong Account View Structure : Usability Test Issue
reacts to
Improve Account View : Usability Test Alternative
Write Argumentation : Recommendation
I agree. Could not find BIC : Usability Test Argumentation
feedback
of
Laura : Prototype Tester
High : Relevance
for
Christine : Prototype Tester
experience
because of
experience
: Confusing Prototype
High : Similarity
: Confusing Prototype
cause
cause
Usability Test of Account View : Application Use
Usability Test of Account View : Application Use
Open Account View : User Interaction
Show Account View : Application Execution
Search Account Holder : User Interaction
Timeout : Application Execution
Open Account View : User Interaction
Show Account View : Application Execution
Search Bank Code : User Interaction
Timeout : Application Execution

Figure 5.17: Example of how PORTNEUF recommends to `argument` for a `usability test alternative` during early design.

developer role specific to software testing. Testers document test executions in *test incident reports*, and in case of a failure, describe how the observed results deviate from the expected. Test incident reports are collected in the *test report summary*, which allows developers to review the failures, prioritize them, and plan for changes in the system. Software testing is recently undergoing changes, which are particularly fostered by the ubiquitous internet.

For example, software beta tests are more and more performed in the public. Practically any person thus can become a system tester, delivering important information about software usage and latent errors to developers. This trend culminates in the phenomenon of the *perpetual beta* [215], where a product is incrementally improved and enhanced while already being rolled out. This practice was originally initiated by Google and has become mainstream especially for web applications.

Another considerable change occurred in the way mobile applications are tested. Mobile testing platforms such as TestFlight [271] or HockeyApp [29] allow developers to distribute test versions of their mobile software to selected persons, who can download and use it with few clicks. These online services use mobile content provisioning techniques to enable wireless ad hoc distribution of the system under test. Moreover, they allow testers to provide feedback and also offer features to analyze test results. The trend is clearly going towards tests with increasing amounts of testers. However, an increasing number of testers leads to several challenges, particularly for test documentation and analysis.

PORTNEUF provides support for two major problems of system tests by a large set of users. First, it enables a structured collection of test results and test reports. PORTNEUF captures the testers' experience with the software while testing and generates recommendations for existing test reports when testers begin to document their results. Second, it allows developers to quantify the test results by calculating the impact of reported test incidents among all testers.

Figure 5.18 shows how the PORTNEUF model can be used during system testing. In this example, `system testers` might discover a `failure` during a system test. Consequently, `test report summaries` correspond to collective user feedback and contain `test incident reports` or `comments`. In the following we describe a scenario that shows how PORTNEUF supports system testing.

`System tester Evelyn` performs a `system test` of the `XML Export` feature in a spreadsheet software. Her task for this test is to save a specific worksheet in XML format under a given file name. To this end, she clicks the `Save as XML` menu item in the spreadsheet, and specifies a file name. Unfortunately, the system generates a `TransformerException`. `Evelyn` closes the message and `retries` the export feature, but the system shows the same `TransformerException` message again. `Evelyn` therefore considers the `system test` to have `failed`. Consequently, she writes a `test incident report`, describing that the `XML Export` is not working. Three hours later, `system tester Natalie`

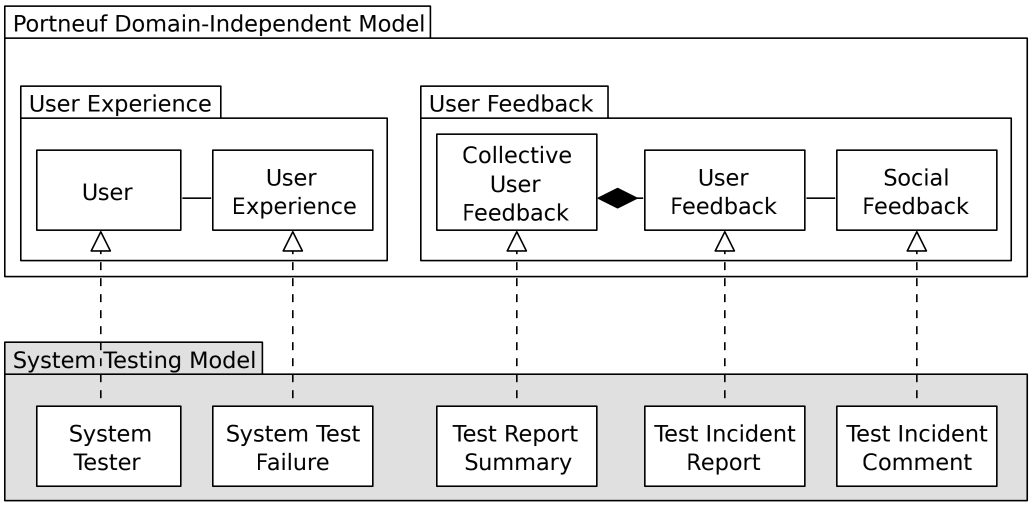


Figure 5.18: Using Portneuf during system testing.

performs the same `system test` of the `XML Export` feature. After opening the spreadsheet, she clicks the `Save as XML` menu item, and specifies a file name. As in the case of `Evelyn`, the system replies with a `TransformerException`. `Natalie` suspects the file name of being the error cause. Consequently, she repeats the export process, but `changes` the target `file name`. Indeed, this time the export is `successful` and the system `writes` the `success` status to a `log` file. At the end of the system test, `Natalie` plans to write a new `test incident report`, but instead obtains a `recommendation` to write a `comment` on the one filed by `Evelyn`, who had a similar experience with the `XML Export` feature. `Natalie` reads `Evelyn`'s `report` and decides to add her experience how she managed to save the file successfully and her assumption on the possible error cause as a comment.

Figure 5.19 depicts the resulting objects. The left side of the figure shows `Evelyn`'s `user experience` and `user feedback`, while the right side illustrates these objects for the case of `Natalie`. Note that in this case, two different test results were obtained by two system testers, what typically complicates a reactive consolidation of such feedback.

### 5.2.3 Software Evolution

Software evolution – also referred to as software maintenance [62] – covers all software engineering aspects of an iteratively and incrementally *evolving* system [113]. It includes activities particularly related to the transition between different iterations, i.e. it is concerned with improving, adapting, and perfecting software and managing changes. Software evolution and support services constitute a growing percentage of the software market, in particular since recent business

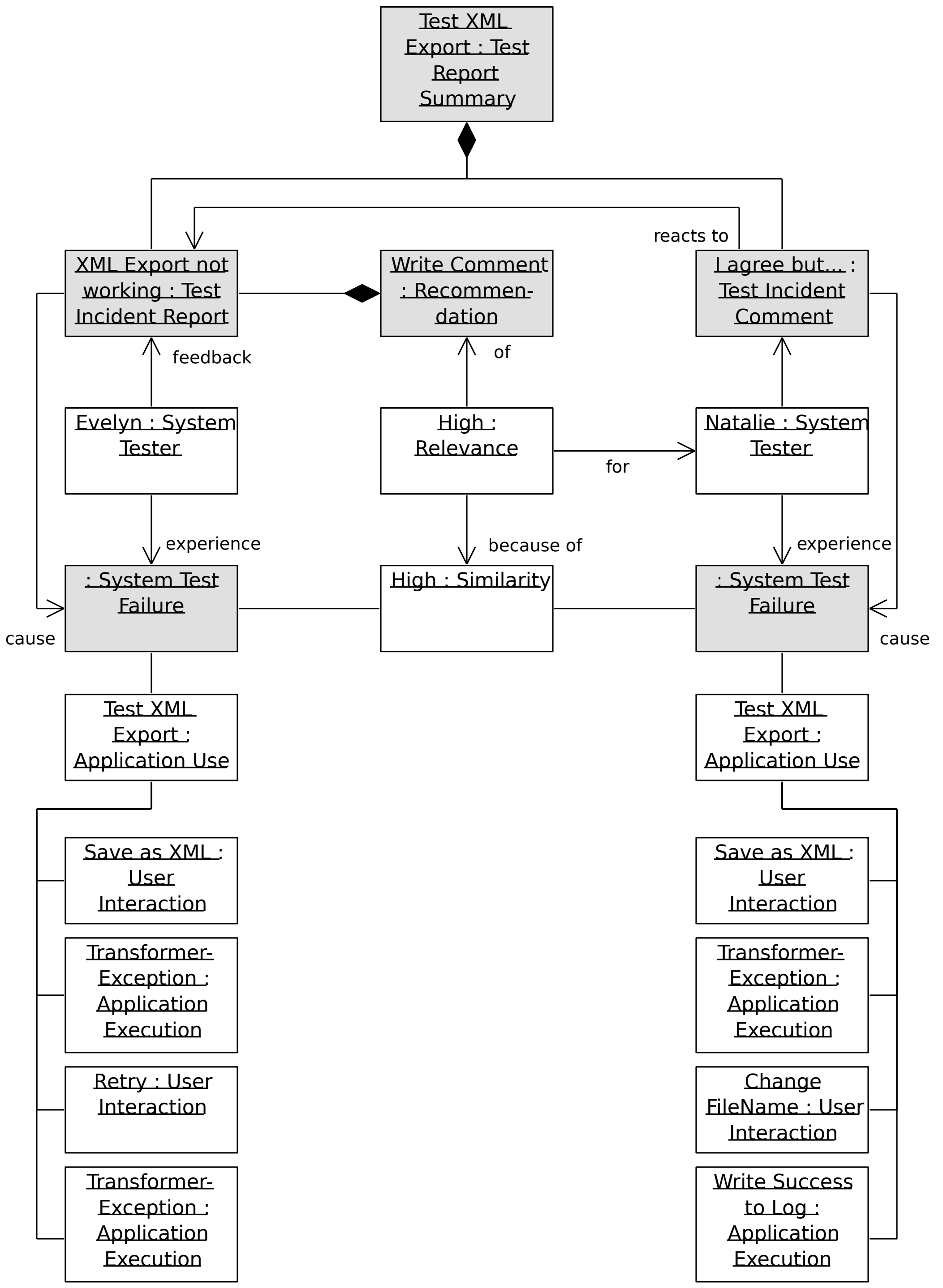

Test XML Export : Test Report Summary
XML Export not working : Test Incident Report
Write Comment : Recommendation
I agree but... : Test Incident Comment
reacts to
feedback
of
Evelyn : System Tester
High : Relevance
Natalie : System Tester
for
experience
because of
experience
: System Test Failure
High : Similarity
: System Test Failure
cause
cause
Test XML Export : Application Use
Test XML Export : Application Use
Save as XML : User Interaction
Save as XML : User Interaction
Transformer-Exception : Application Execution
Transformer-Exception : Application Execution
Retry : User Interaction
Change FileName : User Interaction
Transformer-Exception : Application Execution
Write Success to Log : Application Execution

Figure 5.19: Example of how PORTNEUF recommends to `comment` a `test incident report` during system testing.

models are increasingly focusing on product quality, changing orientation from license-centered to service-centered [219]. The annual costs of software product evolution make between 15 and 25% of the software license list prices [83]. Open source software licenses are often even free of charge, while services offered around the software represent the major source of income. Software evolution is tedious, time-consuming, and intensively involves highly qualified personnel. Studies have shown that software engineers spend a significant amount of their time on maintenance tasks [180]. Assuming a typical software life cycle, more than 75% of the costs are associated to software evolution, and this trend is expected to increase further [88].

As we found in Chapter 3, post-deployment user feedback such as feature requests and bug reports become increasingly important to developers. They analyze user feedback in order to create and prioritize software evolution tasks. In particular the feedback *impact*, i.e. the *frequency* of its occurrence, determines the priority of these tasks. In order to understand the impact of user feedback, developers need to estimate how many users are affected. But this estimation is complicated and requires high effort, when done manually. With an increasing number of users and growing feedback volume, as is the case for novel application distribution platforms (see Chapter 4), a manual analysis becomes actually infeasible.

PORTNEUF provides support for two major problems of software evolution with a large number of users. First, it enables a structured collection of user feedback such as feature requests and error reports. PORTNEUF captures users' experience with the software and creates recommendations for existing reports if users decide to provide feedback. Second, it allows developers to quantify gathered user feedback by calculating the impact of the reported errors or requested features across the user community.

Figure 5.20 shows how we use the PORTNEUF model for user involvement during software evolution. In this example, `end users` might discover that an `application` is `not working correctly` while using it. We model `feedback threads` corresponding to collective user feedback. These contain, for instance, `error reports` or `votings`. In the following we describe a scenario that shows how PORTNEUF supports software evolution.

`End user Edoardo` needs to find a document on his newly installed Mac OS X device, and uses the `spotlight search`. He triggers the `spotlight search` feature using a keyboard shortcut and starts typing. However, the `spotlight results` list remains `empty`. `Edoardo` decides to give the system a little bit more time and `waits` for the results to appear. After one minute, the system `hides` the `result list` without having shown any hit. `Edoardo` is not satisfied with the `user experience` of the new OS X version and decides to write a corresponding `error report` describing that `spotlight search` does not work. Four weeks later, `end user Filippo` who has just updated to the newest version

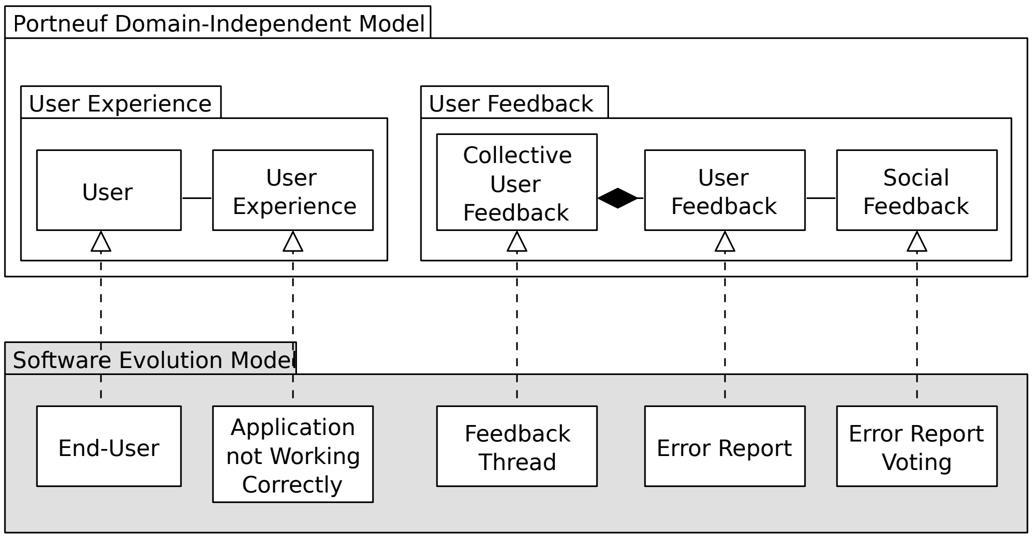


Figure 5.20: Using PORTNEUF during software evolution.

of OS X searches for a text document on his machine. As `Edoardo`, `Filippo` tries to find it via the `spotlight search` feature. He opens the `spotlight search` by clicking the corresponding icon and starts typing. But as in the case of `Edoardo`, the `spotlight results` list remains `empty`. Since `Filippo` is in a hurry he decides to `cancel` the search and tries to find the file manually instead. Consequently, the system `hides` the `result list` without having shown any hit. After eventually having found the document, `Filippo` wants to let Apple know that he is disappointed with the `user experience` of the new OS X version and opens the feedback system in order to write a new `error report`. But instead the system `recommends Filippo` to `vote for` the `error report` written earlier by `Edoardo`, who had a similar experience with the `spotlight search` feature. `Filippo` reads `Edoardo`'s `report` and finds that `Edoardo` has described exactly what he encountered. He therefore `votes` for the error report thus signalizing his agreement.

Figure 5.21 depicts the resulting objects. The left side of the figure shows `Edoardo`'s `user experience` and `user feedback`, while the right side illustrates the corresponding objects for the case of `Filippo`. Note that in this case, two end users have encountered the same error independently from each other. Because end users typically do not browse bug trackers before reporting an error, this scenario usually leads to duplicate error reports which considerably complicate the impact analysis of user feedback.

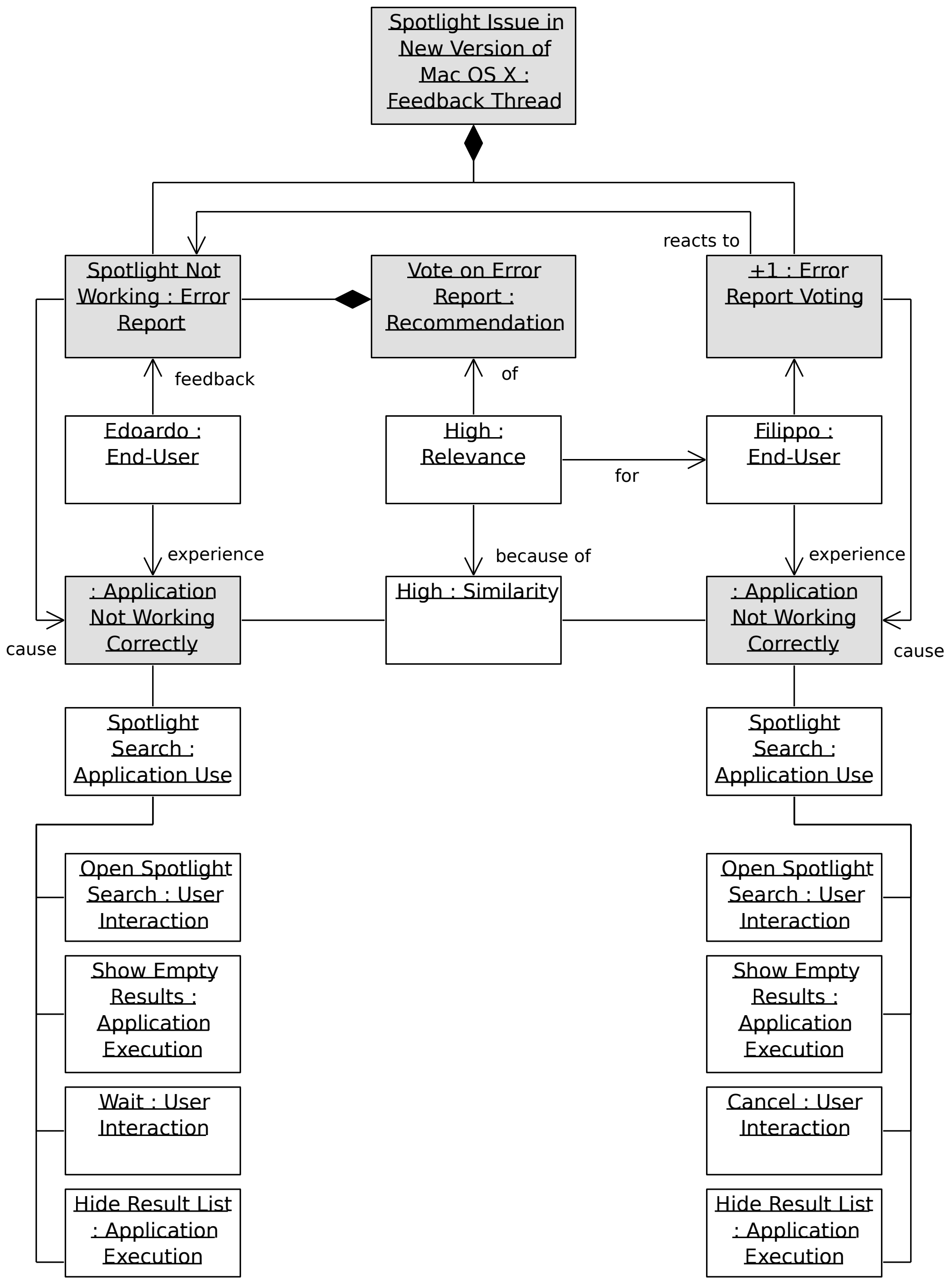

Spotlight Issue in New Version of Mac OS X : Feedback Thread
reacts to
Spotlight Not Working : Error Report
Vote on Error Report : Recommendation
+1 : Error Report Voting
feedback
of
Edoardo : End-User
High : Relevance
for
Filippo : End-User
experience
because of
experience
: Application Not Working Correctly
High : Similarity
: Application Not Working Correctly
cause
cause
Spotlight Search : Application Use
Spotlight Search : Application Use
Open Spotlight Search : User Interaction
Show Empty Results : Application Execution
Wait : User Interaction
Hide Result List : Application Execution
Open Spotlight Search : User Interaction
Show Empty Results : Application Execution
Cancel : User Interaction
Hide Result List : Application Execution

Figure 5.21: Example of how PORTNEUF recommends to `vote` for an `error report` during software evolution.

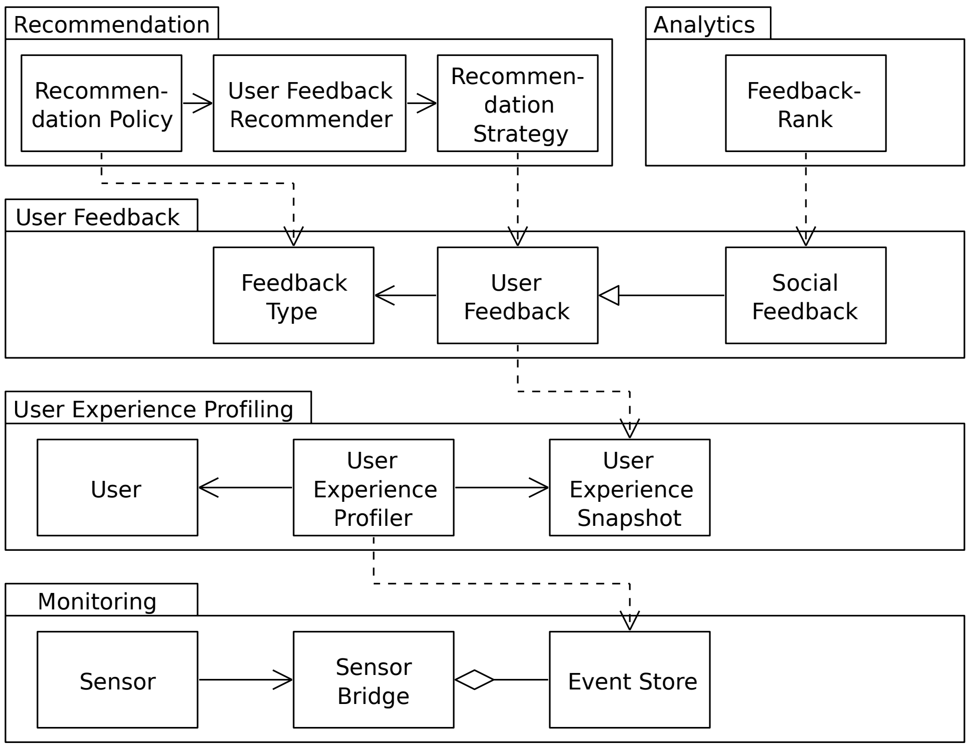


Figure 5.22: Portneuf framework architecture.

## 5.3 Framework Architecture

In this section, we describe the Portneuf framework, an implementation of the domain-independent Portneuf model. Figure 5.22 illustrates the Portneuf framework architecture. The monitoring subsystem (Section 5.3.1) uses sensors to observe the use context and thus is the main enabler of Portneuf's context awareness. The user experience profiling subsystem (Section 5.3.2) aggregates the raw monitored information into user experience snapshots which represent a user's experience with the application in a specific time interval, and allows for a comparison between users and their experience. The user feedback subsystem (Section 5.3.3) accepts feedback provided by the user and combines it with the corresponding user experience snapshot to obtain context sensitive feedback, which is then stored. The two subsystems which correspond to the two main applications of Portneuf are shown on the highest layer. First, the recommendation subsystem (Section 5.3.4) includes a recommender system for user feedback and employs two recommendation strategies to suggest users to rate, vote for, or comment on existing feedback, fostering collective user feedback. Second, the analytics subsystem (Section 5.3.5) implements a ranking algorithm for collective user feedback which utilizes social feedback by the user community in order to rank the feedback obtained for developers.

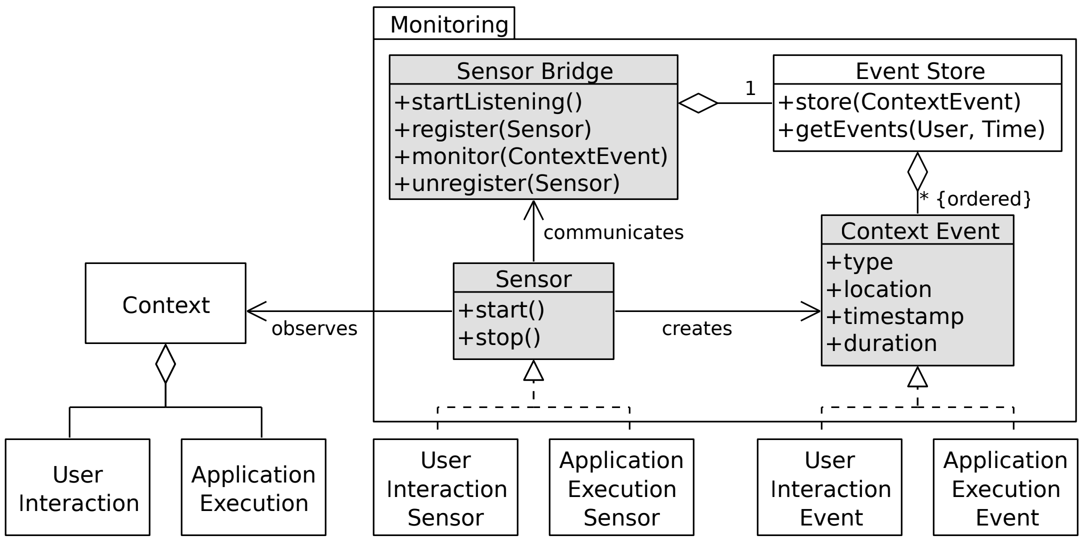


Figure 5.23: PORTNEUF monitoring subsystem.

## 5.3.1 Monitoring

In order to gather information about users' experience, PORTNEUF collects data about their use context, specifically about user interaction and application execution. Figure 5.23 shows the monitoring subsystem of PORTNEUF, which realizes this observation functionality.

Multiple `sensors` monitor the use `context` and create `context events` when they observe specific situations or values. For instance, a `user interaction sensor` might observe how the user moves the cursor, which buttons she clicks, or which text she enters in a form, while an `application execution sensor` might discover exceptions raised or even monitor which methods were called during the execution. `Context events` have a specific `type`, which serves as semantic description of the observed situation. For instance, a `ButtonClickedEvent` denotes that the user clicked a button. The `location` of a `context event` refers to the place where the real observed event has happened. For instance, when a user clicks a button the `location` encapsulates the position of the button in terms of the user interface hierarchy. Furthermore, each context event holds a `timestamp` which denotes when the event has happened, and stores the `duration` of the event. After their creation, `context events` are stored in a sequential order in an `event store`. This data base represents the repository for all further analysis of context events.

Note that we designed PORTNEUF under the *open world assumption* [251]. This means that whether an observation is made or not depends on the presence of a suitable sensor. For instance, if no user interaction sensor is present, this does not mean that there are no user interactions, but that they are unknown.

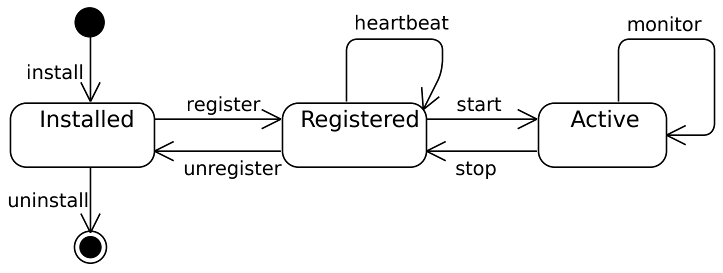


Figure 5.24: PORTNEUF sensor lifecycle.

But since PORTNEUF is based on the open world assumption, new sensors can be added, leading to a more precise image of the reality.

In general, sensors can be generic or specialized. Generic sensors are independent of the monitored application, while specialized sensors are tailored to a specific application. Because of the open world assumption and to allow for future extension and application to arbitrary situations, we designed PORTNEUF without constraining the implementation technology of sensors. Hence, sensors might be part of the original application, standalone components, or linked libraries, implemented in arbitrary programming languages. We designed PORTNEUF as a platform which makes it possible to attach sensors like plugins at runtime, independently from the actual development of PORTNEUF.

The `sensor bridge` provides the interface between the PORTNEUF monitoring subsystem and PORTNEUF sensors, which might run outside the runtime environment of PORTNEUF. To enable technology independence, we made two design decisions. First, we provide the sensor bridge interface using a platform and application independent communication technology. To this end, we implemented a set of REST [100] services. Second, we defined a sensor lifecycle (see Figure 5.24) which all sensors have to implement in order to be PORTNEUF compliant, i.e. to be integratable into the platform.

This lifecycle specifies that PORTNEUF sensors are first `installed` in the user's system, using installation mechanisms which might be specific to these sensors. Note that the PORTNEUF sensor bridge does not need to run for installing sensors. After installation, sensors can `register` at the PORTNEUF sensor bridge. In the registered state, sensors regularly send `heartbeats` to the PORTNEUF sensor bridge in order to obtain commands which control their further behavior. If the sensor bridge does not respond the heartbeats after a specific time period, sensors return in the `Installed` state. After receiving a `start` command, registered sensors become active. Only then do they start to monitor. Whenever sensors have gathered a new event, they send it to the PORTNEUF sensor bridge. The obtained response then contains the next command for the sensors.

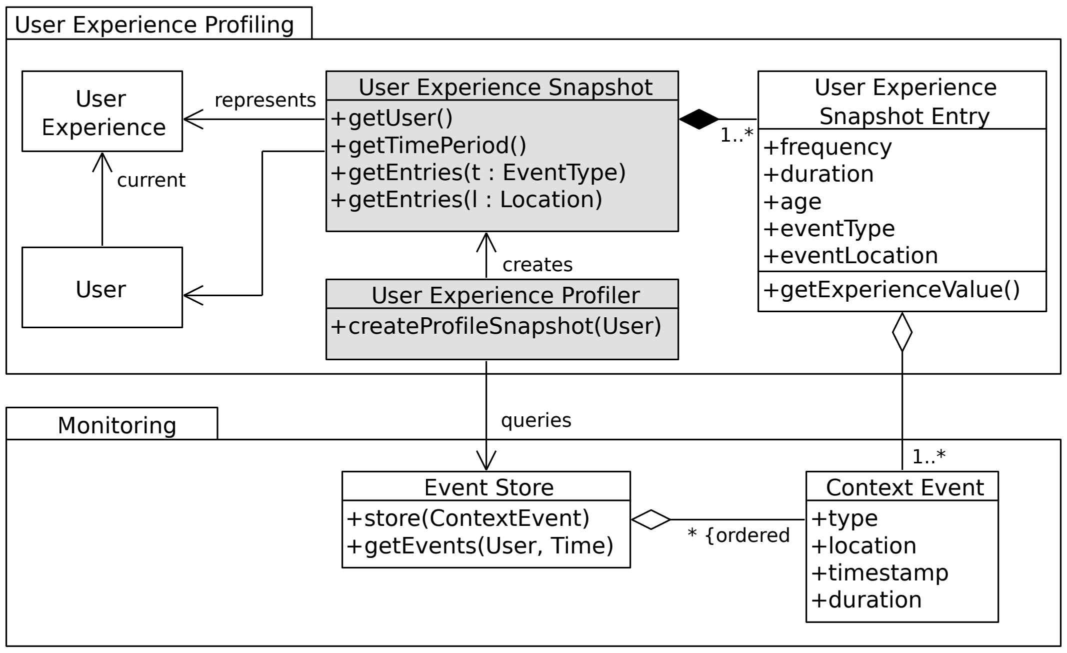


Figure 5.25: PORTNEUF user experience profiling subsystem.

## 5.3.2 User Experience Profiling

As shown in Figure 5.25, the PORTNEUF user experience profiling subsystem provides means for creating `snapshots` which represent the `experience` of a specific `user` for a given `time` period. `User experience snapshots` allow PORTNEUF to compare feedback based on a comparison of the corresponding `user experience` encoded in the feedback. In order to perform this comparison, PORTNEUF first quantifies the experience described in the contained `user experience snapshot entries`. Each `user experience snapshot entry` encodes the `experience` of a `user` with exactly one distinct pair consisting of a `context event type` and a `location`, for instance the `ButtonClickedEvent` event type and the location `viewAccount.btnExport`. User experience is then quantified in form of `frequency`, `duration`, and `age` of the corresponding context events. Research by Maalej [179] has shown that these measures allow for a valid estimation of the relevance of context events.

**Definition 7.** Let $\mathbb{T}$ be the set of context event types and $\mathbb{L}$ the set of context event locations. We define the **experience value** of user $u$ with a context event type $t \in \mathbb{T}$ at location $l \in \mathbb{L}$ formally as

$$expVal_u(t,l) := \frac{freq_u(t,l) \cdot dur_u(t,l)}{age_u(t,l)} \,, \tag{5.1}$$

with $freq_u \geqq 0, dur_u > 0, age_u > 0 \quad \forall t \in \mathbb{T}, l \in \mathbb{L}$.

**Algorithm 5.1** UserExperienceSimilarity.

```
Require: user experience snapshots $ux_{u_1}$, $ux_{u_2}$.
1: $L \leftarrow$ CommonUserExperienceLocations$(ux_{u_1}, ux_{u_2})$
2: $\overrightarrow{exp}_{u_1} \leftarrow$ CalcUserExperienceVector$(ux_{u_1}, L)$
3: $\overrightarrow{exp}_{u_2} \leftarrow$ CalcUserExperienceVector$(ux_{u_2}, L)$
4: $\sigma \leftarrow cosineSim\ (\overrightarrow{exp}_{u_1}, \overrightarrow{exp}_{u_2})$
5: return $\sigma$
```

**Algorithm 5.2** CommonUserExperienceLocations.

```
Require: user experience snapshots $ux_{u_1}$, $ux_{u_2}$.
1: $L \leftarrow \emptyset$
2: for all locations $l \in ux_{u_1}$ do
3:    if $l \in ux_{u_2}$ then
4:       $L \leftarrow L \cup \{l\}$
5:    end if
6: end for
7: return $L$
```

**Corollary.** $\forall t \in \mathbb{T}, l \in \mathbb{L}$, the user experience $ux_u$ of user $u$ can be written as matrix over all experience values as follows:

$$ux_u : \begin{cases} \mathbb{T} \times \mathbb{L} & \rightarrow \mathbb{R}_0^+ \\ (t, l) & \mapsto expVal_u(t, l) \end{cases} \tag{5.2}$$

In the resulting user experience matrix, *column vectors* denote the experience of user $u$ with a specific `location` regarding all `event types` that have occurred. Likewise, *row vectors* capture the user's experience with a certain `event type` regarding all `locations` within the observed context.

Algorithm 5.1 illustrates how we compare two user experience snapshots. In the first step, we obtain their intersection by comparing the contained locations. In the second and third step, we create a user experience *vector* from each matrix

**Algorithm 5.3** CalcUserExperienceVector.

```
Require: user experience snapshot $ux$, set of locations $L$.
1: $\overrightarrow{exp} \leftarrow (0)_{||L||}$ // zero-vector of length $||L||$
2: for $l \in L$ do
3:    $T_l \leftarrow \{t : \exists(t, l) \in ux\}$
4:    $\vec{e_l} \leftarrow ux_{t,l}\ \forall t \in T_l$
5:    $\overrightarrow{exp}_l \leftarrow ||\vec{e_l}||$
6: end for
7: return $\overrightarrow{exp}$
```

$$\underbrace{\begin{array}{l} \\ \text{click} \\ \text{unh. exc.} \end{array}\begin{array}{cc} \text{btnSave} & \text{btnExport} \\ \left(\begin{array}{c} 8 \\ 2 \end{array}\right. & \left.\begin{array}{c} 9 \\ 0 \end{array}\right) \end{array}}_{ux_{u_1}} \qquad \underbrace{\begin{array}{l} \\ \text{click} \\ \text{unh. exc.} \end{array}\begin{array}{cc} \text{btnSave} & \text{btnExport} \\ \left(\begin{array}{c} 8 \\ 3 \end{array}\right. & \left.\begin{array}{c} 9 \\ 9 \end{array}\right) \end{array}}_{ux_{u_2}}$$

$$cosineSim\left(ux_{u_1}\Big|_{\text{btnSave}}, ux_{u_2}\Big|_{\text{btnSave}}\right) = \frac{8 \cdot 8 + 2 \cdot 3}{\sqrt{8^2+2^2} \cdot \sqrt{8^2+3^2}} = 0.994$$

$$cosineSim\left(ux_{u_1}\Big|_{\text{unh.exc.}}, ux_{u_2}\Big|_{\text{unh.exc.}}\right) = \frac{2 \cdot 3 + 0 \cdot 9}{\sqrt{2^2+0^2} \cdot \sqrt{3^2+9^2}} = 0.316$$

$$\textsc{UserExperienceSim}\left(ux_{u_1}, ux_{u_2}\right) = cosineSim\left(\begin{pmatrix}\sqrt{68} \\ 9\end{pmatrix}, \begin{pmatrix}\sqrt{73} \\ \sqrt{162}\end{pmatrix}\right) = 0.989$$

Figure 5.26: Comparing different user experience aspects using cosine similarity.

by calculating the Euclidean norm of all its column vectors. Although user experience matrices are typically sparse, this step is necessary in case multiple event types were monitored at one location. In the last step, we apply the cosine similarity measure [253] to the resulting user experience vectors. Cosine similarity represents an estimation of the angle between two vectors:

**Definition 8.** The **cosine similarity** of two vectors $a, b \in \mathbb{R}^n$ is defined as

$$cosineSim(a,b) := \frac{a \cdot b}{\|a\| \; \|b\|} \tag{5.3}$$

Cosine similarity is a commonly used similarity measure in information retrieval, and has been successfully applied in content-based recommender systems [230]. We apply cosine similarity to compare user experience snapshots since it is well suited to identify commonalities and because it returns a normed result. An alternative to cosine similarity would be the Euclidean distance metric.

Context-aware recommender systems [9] take into account additional contextual information, such as time or location of a user, when recommending items. Portneuf captures the experience of a user with specific locations in an application regarding contextual events. Consequently, apart from a general comparison of users' experience, the Portneuf user experience profile allows for a context-sensitive user comparison based on selected event types or event locations, which may be relevant in the particular context. Hence, Portneuf is able to generate recommendations depending on the specific context.

Figure 5.26 illustrates with a simple example how context-sensitive recommendations can be obtained. It shows the user experience matrices of two users $u_1$ and $u_2$. Both users have a similar experience with the button "btnSave", but user $u_2$ experiences significantly more unhandled exceptions than user $u_1$. If

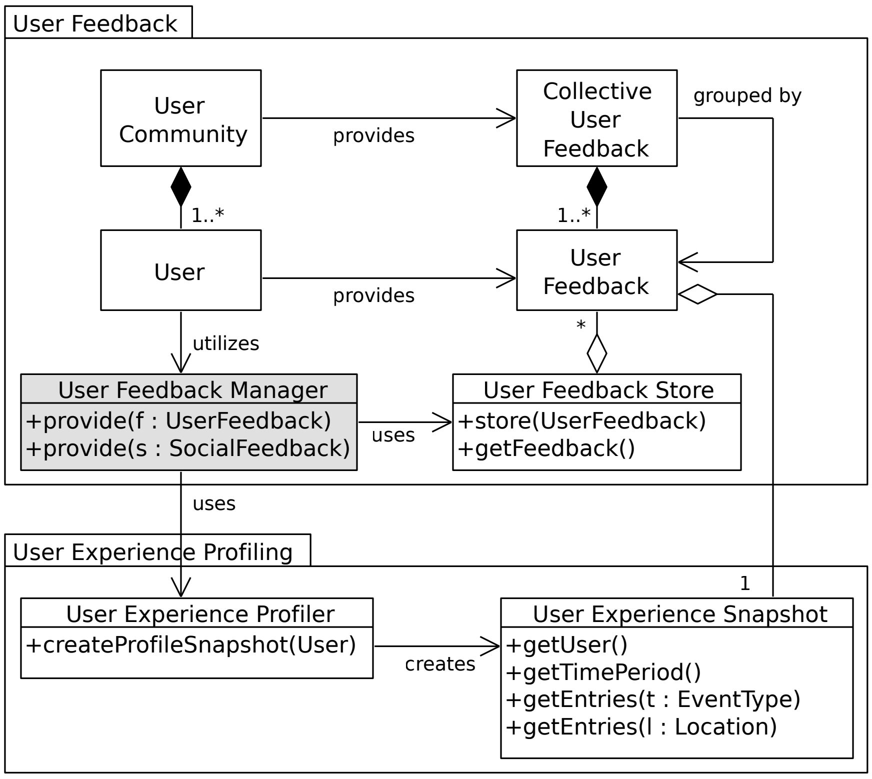


Figure 5.27: Portneuf user feedback subsystem.

user $u_2$ wants to give feedback on the "btnSave" button, the user feedback from user $u_1$ is a good candidate to search for existing feedback. On the other hand, if user $u_2$ wants to report an unhandled exception, the feedback from user $u_1$ is much less relevant.

### 5.3.3 User Feedback

Figure 5.27 shows the class model of the Portneuf user feedback system. Its main purpose is to allow `users` to provide in situ `feedback` while using the application and to link their feedback with the corresponding `snapshot` of their `user experience`. Whenever a user provides feedback to the `user feedback manager`, it queries the `user experience profiler` to create a `user experience snapshot` for the `user` in question. This snapshot represents the `user experience` up to the current point in time. The `user feedback manager` connects the created snapshot with the collected feedback and stores both in the `user feedback store`. It is this combination between provided feedback and underlying user experience which enables Portneuf to generate content-based recommendations. Because `user experience snapshots` contain context events, user feedback collected with Portneuf is *context-sensitive.*

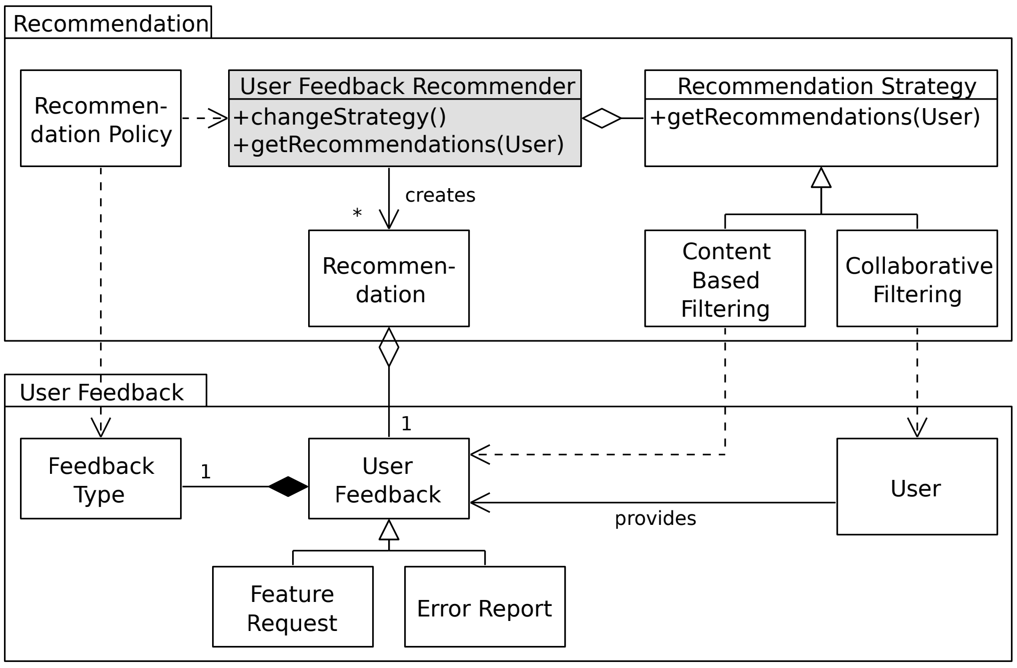


Figure 5.28: PORTNEUF recommendation subsystem.

## 5.3.4 Recommendation

Recommender systems have been successfully applied in various domains such as recommendations for movies [137], financial services [97], websites [229], and software components [241]. We distinguish four basic recommendation strategies. *Collaborative filtering* [137] proposes items based on the information about nearest neighbors: item preferences of users with a similar observed behavior as the current user are exploited to infer recommendations, similar to word-of-mouth promotion. *Content-based filtering* [229] analyzes the observed behavior of the current user and recommends items with characteristics similar to those of items the current user has already rated high. *Knowledge-based* recommender systems [97] rely on knowledge about the user. For instance, a knowledge-based recommender system for a shopping application could rely on the available item assortment as well as marketing and sales knowledge. Finally, *group* recommender systems [147, 193, 213] support human decision making by taking into account factors such as the opinion of other group members, individual motivations, and personal preferences. Their goal is to achieve consensus among the members of a group.

As shown in Figure 5.28, PORTNEUF creates `recommendations` of existing `user feedback`, which is relevant for the user, whenever she is about to provide feedback. In the best case, these recommendations contain the feedback the user wanted to provide. Otherwise, they might include similar feedback, which allows the user to explain her experience by commenting. PORTNEUF implements two complementary recommendation strategies: `content-based filtering` and `collaborative filtering`.

The content-based filtering strategy compares the current user's `experience profile` with `user experience profile`s attached to already existing feedback from other users. It then recommends the user feedback with the most similar `user experience profile`s, which the user has not yet seen.

The collaborative filtering strategy calculates user preferences by analyzing their participation in existing collective user feedback. For instance, users who voted for the same feedback have the same preferences regarding this feedback. The strategy then recommends user feedback which users with similar preferences have reported or voted for, and which the current user has not yet seen.

The `recommendation policy` decides at runtime which strategy is suitable based on the `type` of `feedback` the user is about to provide. The PORTNEUF framework supports two types of user feedback: `error reports` and `feature requests`. We assume that these feedback types require different recommendation strategies, because of their differing semantics. With error reports users refer to events which have *happened*, while feature request describe which functionality is *missing*. We formulate the following hypothesis.

**Hypothesis 5.3.** *Content-based filtering is most suitable to create recommendations for error reports, while collaborative filtering is most suitable to generate recommendations for feature requests.*

In practice, both content-based filtering and collaborative filtering methods have strengths and weaknesses [52]. The best known weakness is probably the "ramp-up" problem [163], a term which refers to two different startup scenarios. In the first scenario, recommendations need to be created for a *new user*. Since calculating recommendations includes comparing a user's preferences to the preferences of other users, users with few or even no expressed preferences are difficult to categorize. The second scenario is similar but refers to *new items*. Items which have not yet been rated by a sufficient number of users are difficult to be recommended. While collaborative filtering is affected in both cases, content-based filtering deals well with new items. However, before recommendations can be generated with content-based filtering, typically models of the users have to be learned. PORTNEUF deals with this requirement by utilizing user experience profiles as *fingerprints* which allow the recommender system to directly look up the most relevant items, independently from the user. Hence, PORTNEUF overcomes the ramp-up problem for new users.

In order to compensate for the remaining weaknesses, PORTNEUF combines both recommendation strategies to form a hybrid recommender system [52]. Then, recommendations are calculated using both strategies and combined following Hypothesis 5.3: Recommendations for error reports are determined by

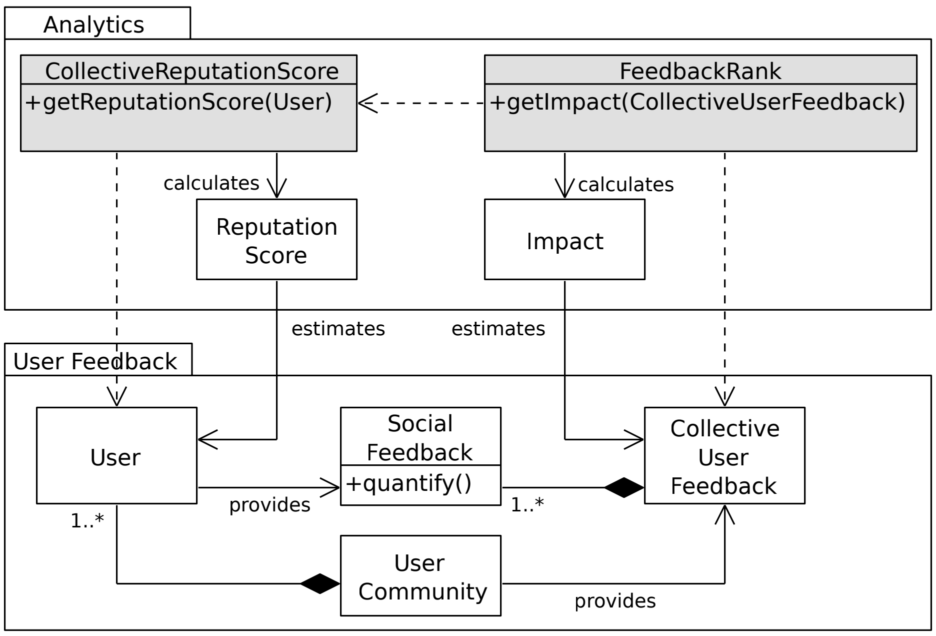


Figure 5.29: Portneuf analytics subsystem.

**Algorithm 5.4** FeedbackRank.

**Require:** collective user feedback *cuf*.

```
1: $\varrho \leftarrow$ CollectiveReputationScore($cuf$.getReporter())
2: for all social feedback $sf \in cuf$ do
3:     $\sigma \leftarrow sf$.quantify()
4:     $\varrho \leftarrow \varrho + \sigma \cdot$ CollectiveReputationScore($sf$.getReporter())
5: end for
6: return $\varrho$
```

weighting content-based filtering recommendations with 80% and collaborative filtering recommendations with 20%, and vice versa for feature requests.

### 5.3.5 Analytics

Figure 5.29 depicts the Portneuf analytics subsystem. Its main purpose is to estimate the `impact` of the gathered `collective user feedback`. For this purpose, we introduce a new algorithm called FeedbackRank, which relies on `social feedback` and `reputation scores` in order to calculate feedback impact. It takes into consideration how much every contributing user is valued by the community.

The FeedbackRank algorithm assigns every `collective user feedback` a score that indicates how important in the `user community` that feedback is. As illustrated in Algorithm 5.4, FeedbackRank estimates the `reputation score`

**Algorithm 5.5** CollectiveReputationScore.

```
Require: user $u$, factor $\epsilon$.
 1: $SV_u \leftarrow u$.getIncomingSocialFeedback()
 2: $\gamma \leftarrow 0$
 3: for all social feedback $sf \in SV_u$ do
 4:     $v \leftarrow sf$.getReporter()
 5:     if $v \neq u$ then
 6:         $SF_v \leftarrow v$.getSocialFeedback()
 7:         $\sigma \leftarrow sf$.quantify()
 8:         $\Gamma \leftarrow$ CollectiveReputationScore($v$)
 9:         if $\Gamma > 0$ then
10:             $\gamma \leftarrow \gamma + \sigma \cdot \Gamma / \|SF_v\|$
11:         end if
12:     end if
13: end for
14: $\gamma \leftarrow \epsilon \cdot \gamma$
15: return $\gamma$
```

of the `collective user feedback`'s original reporter and adds the `quantification` of the included `social feedback`, weighted by the `reputation score` of the respective user who published it.

As described in Section 5.1.4, there are various methods for computing `reputation scores`. Portneuf, uses the flow model based algorithm CollectiveReputationScore (see Algorithm 5.5), which works similar to Google's PageRank [223]. The algorithm calculates the collective reputation score of a user, which is defined as follows:

**Definition 9.** Let $u$ be a user. Then let $SF_u$ be the set of `social feedback` provided by user $u$ and $SV_u$ the set of `social feedback` which was given on $u$'s feedback (called $u$'s *social value*). Let further $n_{sf} = \left\|SF_{author(sf)}\right\|$ be the total amount of `social feedback` given by the author of *sf* and $\varepsilon$ a factor used for normalization. With $|sf|$ we denote the quantification of social feedback *sf* (e.g. +1 for upvote). Then, we define the **collective reputation score** of $u$ as:

$$CRS(u) := \varepsilon \sum_{sf \in SV_u} \frac{|sf| \cdot CRS(sf)}{n_{sf}}$$

The CollectiveReputationScore algorithm assigns each user the values they obtained from other users, weighted by their reputation scores. The collective reputation score of a user is divided evenly among the `social feedback` she provided on other users' feedback to contribute to their reputation score.

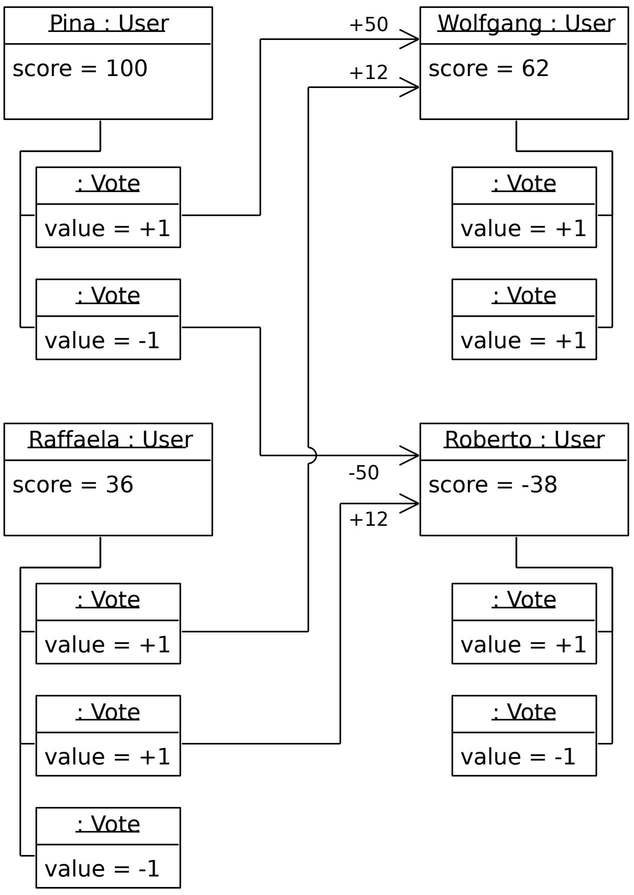


Figure 5.30: Simplified example of CollectiveReputationScore.

Similar to PageRank, CollectiveReputationScore is recursive, but can be calculated starting with any set of reputation scores for all users and iterating the calculation until it converges.

Figure 5.30 illustrates the calculation of the collective reputation score with an example. User `Pina` has a `score` of 100, whose origin is not shown in this figure. She has `voted` for a feedback by user `Wolfgang` and against one by user `Roberto`. Consequently, user `Wolfgang` receives 50 points, while user `Roberto` gets a negative value of $-50$ points. Because `Wolfgang` obtained an additional positive `vote` by user `Raffaela`, who has a `score` of 36, his `score` adds up to 62. Also user `Roberto` received a positive `vote` by `Raffaela`, but in the end his `score` only makes $-38$.

## 5.4 Related Work

We discuss two areas of related work: research dealing with user feedback for software, and existing systems that gather user feedback and capture user experience.

### 5.4.1 User Feedback Research

Schneider et al. [259, 260] propose ConTexter, an approach to collect changing end user requirements during system evolution. To capture the reactions of end users on a system, ConTexter gathers feedback in context with mobile devices. In order to focus the feedback, ConTexter allows system providers to establish filters, which should guide end users when providing feedback. The problem Schneider et al. address is similar to the one PORTNEUF deals with. However, their solution differs from PORTNEUF in three major aspects. First, feedback collected with ConTexter is anonymous, while in PORTNEUF the public visibility of user feedback is the main enabler for the consolidation of feedback. Second, in PORTNEUF the user community consolidates their own feedback using comments, ratings, and votings, while in ConTexter the system provider defines filters with which users may tag their feedback. Third, ConTexter is designed to support users of IT-ecosystems, such as airports or universities, while PORTNEUF targets end users of software applications in general. Therefore, ConTexter aims at capturing the physical context, allowing users to attach audio or video recordings, while PORTNEUF focuses on events initiated by the user (user interactions) or the software itself (application execution).

In an additional work, Gärtner and Schneider [109] propose a heuristic function in order to prioritize end user feedback. It takes into account quantifiable features of feedback text and attachments that significantly influence its importance, while trying to avoid the incorporation of complex domain knowledge. In a first evaluation, the authors could show that this heuristic yields helpful estimates of feedback priority. Instead of using heuristic functions, the PORTNEUF model enables the utilization of the user community's collective intelligence to prioritize feedback. The advantage of our approach is the obtained feedback structure which also avoids duplicates.

Chilana et al. [66] present a tool called "LemonAid" that allows users to search for help on web pages by selecting a UI element (e.g. label, link, or image) of which they believe that it is relevant to their problem. In an initial study the authors could show the feasibility of relating questions and answers based on UI elements, which are essentially part of the context of use. This supports our assumption that the context of use is helpful to estimate the relevance of user feedback. The goals of LemonAid and PORTNEUF are similar. LemonAid aims at connecting users who seek help with users who provide help, while PORTNEUF tries to connect users who made similar experience with an application.

However, there are three major differences. First, PORTNEUF does not rely on a user-based estimation of relevant content, but calculates relevance based on an automatically gathered measure of user experience. Second, while LemonAid only considers UI elements, we propose a more generic model, which can be tailored to different applications and user groups. Third, LemonAid can only provide help for UI related problems, while PORTNEUF is able to link users in arbitrary yet similar situations.

Froehlich et al. [107] propose a framework called "MyExperience" to gather feedback on users' experience with mobile phones. To this end, the authors combine passive logging of device usage, environmental sensor readings, and active context-triggered user experience sampling. The results showed that the system could help researchers or service vendors to better understand usage contexts. Similarly, we claim that the use context provides means to capture user experience, but in contrast to MyExperience, the PORTNEUF model aims at supporting the impact analysis of feedback.

Özçelik Buskermolen et al. [216] explore the types of end user information considered useful by designers for early concept evaluations. The results indicate that designers prefer elaborate feedback which indicates clear attitudes and motivations as well as feedback that reveals users' past experiences. This supports our approach to capture user experience and utilize it as grouping construct for collective user feedback.

Ali et al. [8] suggest to design adaptive systems capable of "social adaptation". The term refers to a system's ability to analyze user feedback and react to it by changing its behavior, with the goal of always meeting users' requirements correctly and efficiently. To this end, the authors assume user feedback which is not natural language based and therefore interpretable by the adaptive system. Although this scenario is different from the one that drives our research, the underlying assumption is the same: the wisdom of crowds can be exploited to improve and evolve a system over time. Additionally, the notion of adaptive systems leads to an interesting question, namely how much control do developers need when reacting to user feedback? In an extreme case, PORTNEUF could be utilized to enable social adaptation based on common, natural language feedback.

### 5.4.2 User Feedback Systems

In recent years, systems capturing user experience and software usage and allowing users to provide feedback have gained attention in practice, especially for web sites. Several commercial services such as Usabilia[4], AppTelemetry[5],

[4]http://usabilla.com
[5]http://www.apptelemetry.com/en/feedback-feature.html

or HeadsUp! User Engagement[6] sell features which analyze how users move through web pages and collect their feedback.

Social media enabled platforms like UserVoice [277] and GetSatisfaction [110] allow users to collaboratively share new ideas and vote on existing suggestions for new features. Bajic and Lyons [15] found that this collaboration focuses users' efforts and leads to more homogeneous feature requests. UserVoice and GetSatisfaction even try to avoid duplicate feedback by analyzing the text written by a user while she is providing feedback. However, this approach is not equal to ours because of three reasons.

First, the comparison of user feedback is based on a syntactic natural language analysis. However, natural language is highly ambiguous and different users may therefore describe the same experience using different words (e.g. "button" instead of "link", "click" instead of "tap", or "UI" or "form" instead of "view"). Second, these platforms do not capture user experience. But research has shown that feedback written by users typically lacks this information [298]. The user experience collected by PORTNEUF can be symbolized and might be helpful for developers in order to better understand the provided feedback. Third, the platforms are not context-aware. This means that all user feedback is considered equal in all situations. On the contrary, PORTNEUF allows for weighting particular user feedback more than other based on the particular context of use.

## 5.5 Summary

We have introduced PORTNEUF, a domain-independent model which supports continuous user involvement by enabling proactive and context-aware recommendations of user feedback based on users' experience as follows:

- PORTNEUF defines and conceptualizes *user experience*. Because it is not directly measurable and subjective when described by users, we summarize factors which influence user experience and hence enable an indirect measurement. In particular, the application use history and use context belong to the major factors.

- The presented model also includes *user feedback* and relates it to user experience. It defines *collective user feedback*, which groups feedback of multiple different users around common experience. Collective user feedback captures the opinion of the user community as a whole and enables a quantification of its impact.

- PORTNEUF derives the relevance of specific feedback for a particular user in a given context by a comparison of the different user experiences. Based on this relevance, PORTNEUF generates *recommendations* of existing user

[6]http://www.headsupuserengagement.com

feedback in order to proactively group different users' feedback around a common topic and to avoid the creation of duplicates.

- The resulting collective user feedback includes the transitive closure of all related feedback, and thus provides means for quantifying the corresponding *impact* of the described issue for the user community.

We have shown how the PORTNEUF model can be applied during three software engineering activities, *early design*, *system testing*, and *software evolution*, and instantiated it exemplarily for each of them. Moreover, we substantiated the application of PORTNEUF to software evolution – the main focus of this dissertation – by describing a software framework that implements the PORTNEUF model for software evolution. The framework solves the problems of developers which we have discovered in Chapter 3 following a proactive approach. It automatically captures users' experience and creates feedback recommendations for users based on the similarity of their experiences. Finally, it allows developers to assess the impact of user feedback in a quantitative way. In the next chapter, we describe the formative and summative evaluation of the PORTNEUF framework.

# Chapter 6

# Evaluation

*«It is entirely possible that behind the perception of our senses, worlds are hidden of which we are unaware.»*

— Albert Einstein

In the previous chapters, we described PORTNEUF, a proactive and context-aware framework which groups user feedback according to users' prior experience with an application. We described how PORTNEUF recommends existing, relevant feedback to users to rate, vote for, or comment on, and showed how it allows developers to assess the collective value of user feedback with the FEEDBACKRANK metric.

In this chapter, we describe the evaluation of our solution. In Section 6.1 we show the applicability and feasibility of our concepts. We prototypically implemented the PORTNEUF framework for two different applications. In Section 6.2 we describe our evaluation setting, including questions, methodology, and data. Section 6.3 summarizes the results of the evaluation. We describe how users provided feedback with PORTNEUF and illustrate the resulting effects on developers' work with user feedback. We quantify the effects of PORTNEUF on the amount of duplicate user feedback, describe how it facilitates impact assessment, and collect emerging issues and improvement suggestions. Finally, Section 6.4 summarizes our findings.

## 6.1 Framework Implementation

The evaluation of PORTNEUF consisted of two parts, a formative evaluation and a summative evaluation. The goal of the formative evaluation was to test the feasibility of PORTNEUF and to explore different recommendation strategies in a controlled environment. The obtained results allowed us to improve the framework before using it in a real-world setting. The goal of the final summative evaluation was to analyze the user feedback collected by PORTNEUF for an established application with a large user community.

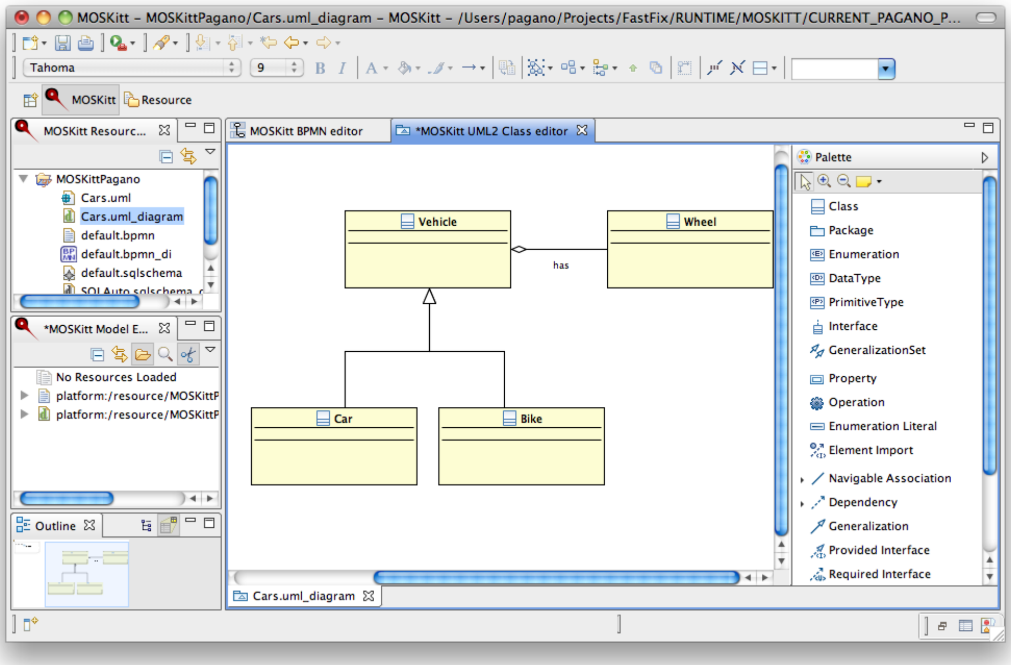


Figure 6.1: MOSKitt UML modeling tool.

### 6.1.1 Formative Evaluation

For the formative evaluation, we a implemented a vertical prototype of PORTNEUF within the FastFix project[1]. The overall goal of FastFix is to provide software engineers with a "real-time" maintenance environment which increases efficiency and reduces total cost. It aims at improving accuracy in identification of failure causes and facilitating their resolution. The FastFix platform includes a set of software tools to remotely monitor user environments which are able to collect contextual information on application execution and user interaction [219].

Prototype development consisted of two steps, the implementation of the PORTNEUF framework for the FastFix platform, and the integration of the user feedback functionality in a FastFix target application. All PORTNEUF framework components except the monitoring subsystem can run separately from the application, and are accessible for all users. The PORTNEUF monitoring subsystem needs to run on the same machine as the application for which feedback should be collected. We developed the PORTNEUF prototype in Java using OSGi and Eclipse Rich Client Platform (RCP). The PORTNEUF user feedback functionality was integrated in Prodevelop MOSKitt[2], a UML modeling tool also

[1]http://fastfix-project.eu
[2]http://www.moskitt.org/eng

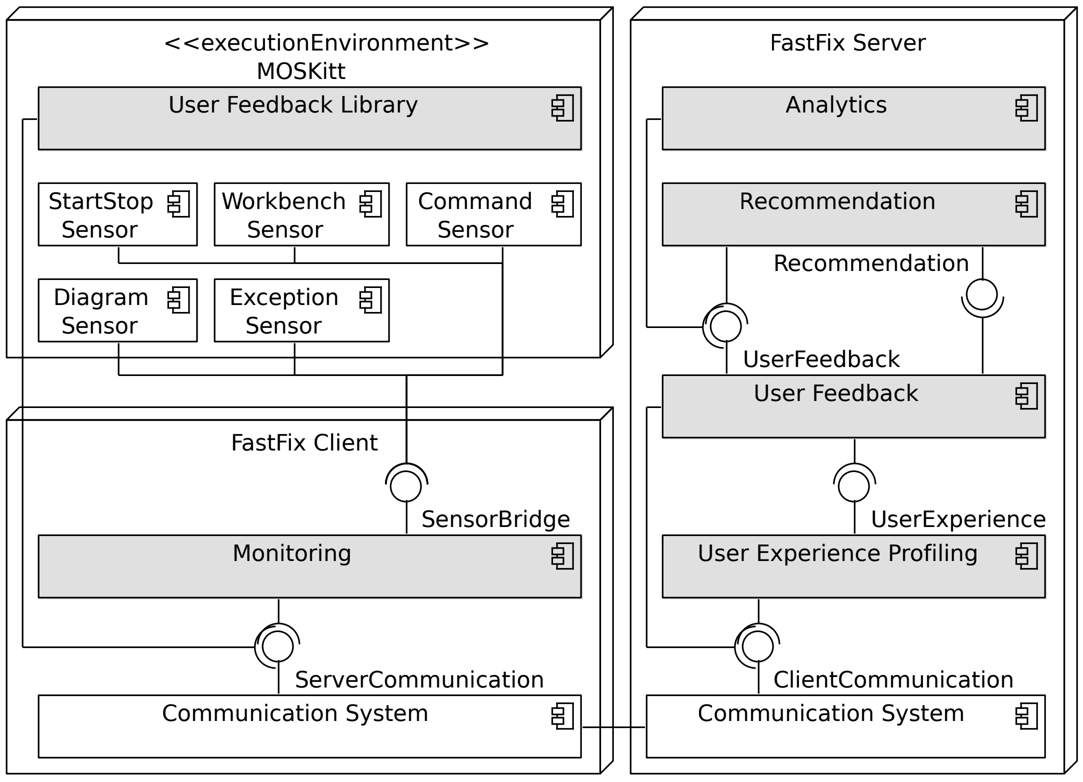


Figure 6.2: Deployment of the PORTNEUF framework for MOSKitt. PORTNEUF components are shown in gray.

based on Eclipse RCP technology. Figure 6.1 depicts the MOSKitt user interface.

Figure 6.2 illustrates how the PORTNEUF framework components are distributed between the target application MOSKitt and the FastFix client and server. To monitor the application use context, the prototype uses sensors developed within FastFix. These sensors transfer observed events to the monitoring component which we integrated in the FastFix client, a standalone proxy application to collect local user data and send it to the FastFix server, which represents the common back end. The remaining PORTNEUF components are integrated in the FastFix server. Both sensors and user feedback library are deployed via so called update sites – a plugin mechanism for RCP applications which facilitates deployment and updates. As a result, both components can be used not only with MOSKitt, but with arbitrary RCP applications.

Figure 6.3 illustrates the seamless integration of PORTNEUF into the MOSKitt user interface. Users can provide feedback by selecting the corresponding menu item. When users decide to provide feedback, they obtain recommendations of existing user feedback which they can vote for or comment on. Figure 6.4 illus-

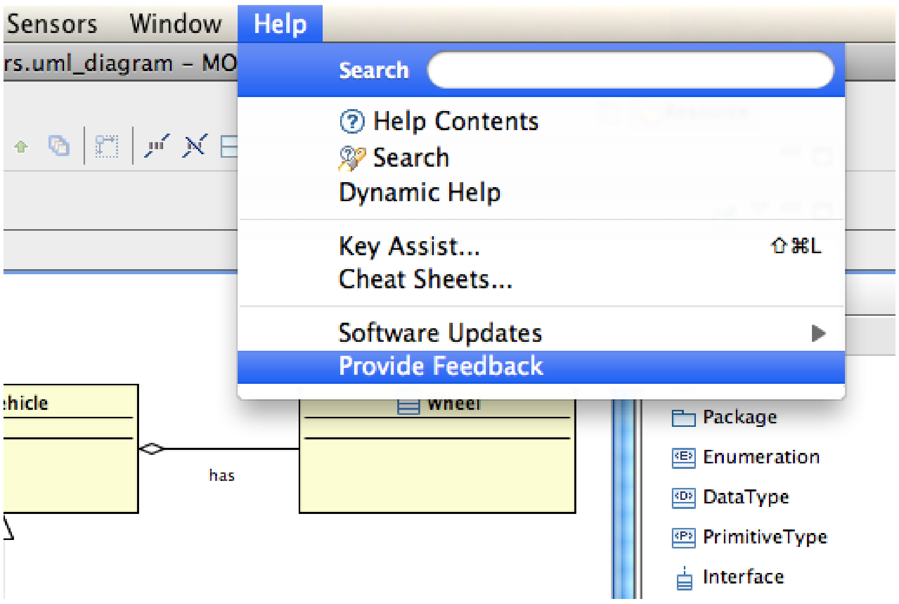


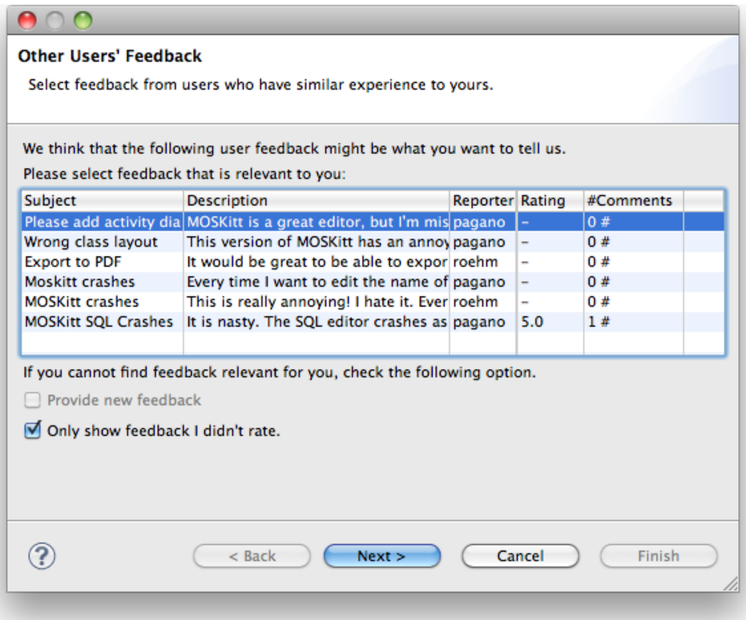
Figure 6.3: Integration of PORTNEUF in MOSKitt.

trates the generated user feedback recommendations and exemplifies collective user feedback.

The formative evaluation showed that PORTNEUF is feasible, as we were able to implement the framework as specified, to integrate it in an existing real-world application, and to generate recommendations for existing user feedback. In addition, we made the following two observations, which allowed us to create an improved version of the framework for the summative evaluation.

First, when generating user feedback recommendations, we observed that the content-based recommendation strategy performed generally better than the collaborative filtering strategy. We found that this result was due to the low number of users who had tested the prototype, because the low amount of social feedback they had provided was not sufficient to build user profiles. Since we expected similar results in particular at the beginning of the summative evaluation, we ensured that PORTNEUF created content-based recommendations as fallback solution for feature requests. In addition, we made the hybrid recommendation policy adjustable, so that the weight of recommendations obtained by the content-based and collaborative filtering strategies could be modified if needed.

Second, we tested the collaborative filtering strategy with two different metrics to calculate user similarity: the Euclidean norm and the Pearson correlation coefficient [243]. While both metrics measure similarity, the Pearson coefficient tends to give better results when the compared data is not normalized. Nevertheless, we obtained better recommendations with the Euclidean norm, and consequently used this metric for the summative evaluation.


Other Users' Feedback

Select feedback from users who have similar experience to yours.

We think that the following user feedback might be what you want to tell us.

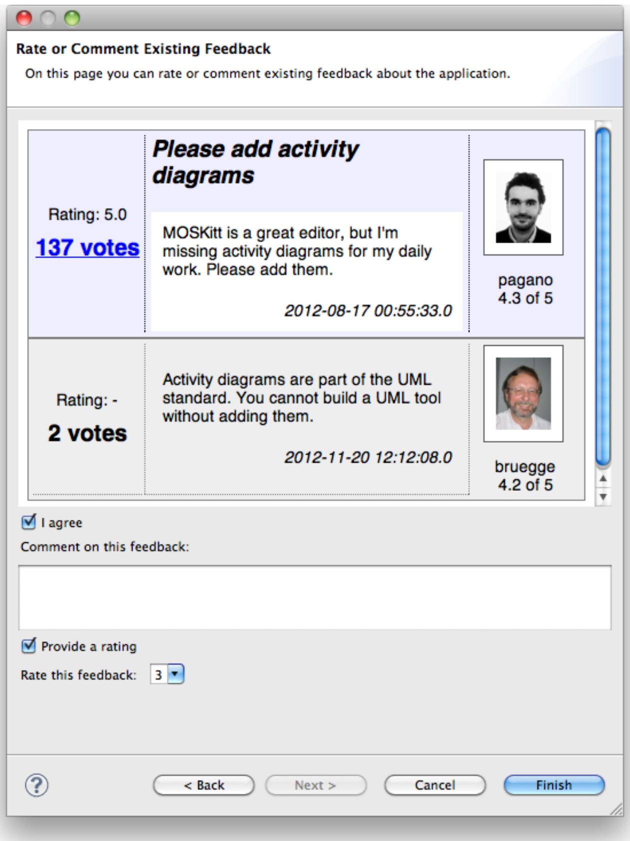
Please select feedback that is relevant to you:

| Subject | Description | Reporter | Rating | #Comments |
|---|---|---|---|---|
| Please add activity dia | MOSKitt is a great editor, but I'm mis | pagano | - | 0 # |
| Wrong class layout | This version of MOSKitt has an annoy | pagano | - | 0 # |
| Export to PDF | It would be great to be able to expor | roehm | - | 0 # |
| Moskitt crashes | Every time I want to edit the name of | pagano | - | 0 # |
| MOSKitt crashes | This is really annoying! I hate it. Ever | roehm | - | 0 # |
| MOSKitt SQL Crashes | It is nasty. The SQL editor crashes as | pagano | 5.0 | 1 # |

If you cannot find feedback relevant for you, check the following option.

☐ Provide new feedback

☑ Only show feedback I didn't rate.


Rate or Comment Existing Feedback

On this page you can rate or comment existing feedback about the application.

Rating: 5.0
137 votes

Please add activity diagrams

MOSKitt is a great editor, but I'm missing activity diagrams for my daily work. Please add them.

2012-08-17 00:55:33.0

pagano
4.3 of 5

Rating: -
2 votes

Activity diagrams are part of the UML standard. You cannot build a UML tool without adding them.

2012-11-20 12:12:08.0

bruegge
4.2 of 5

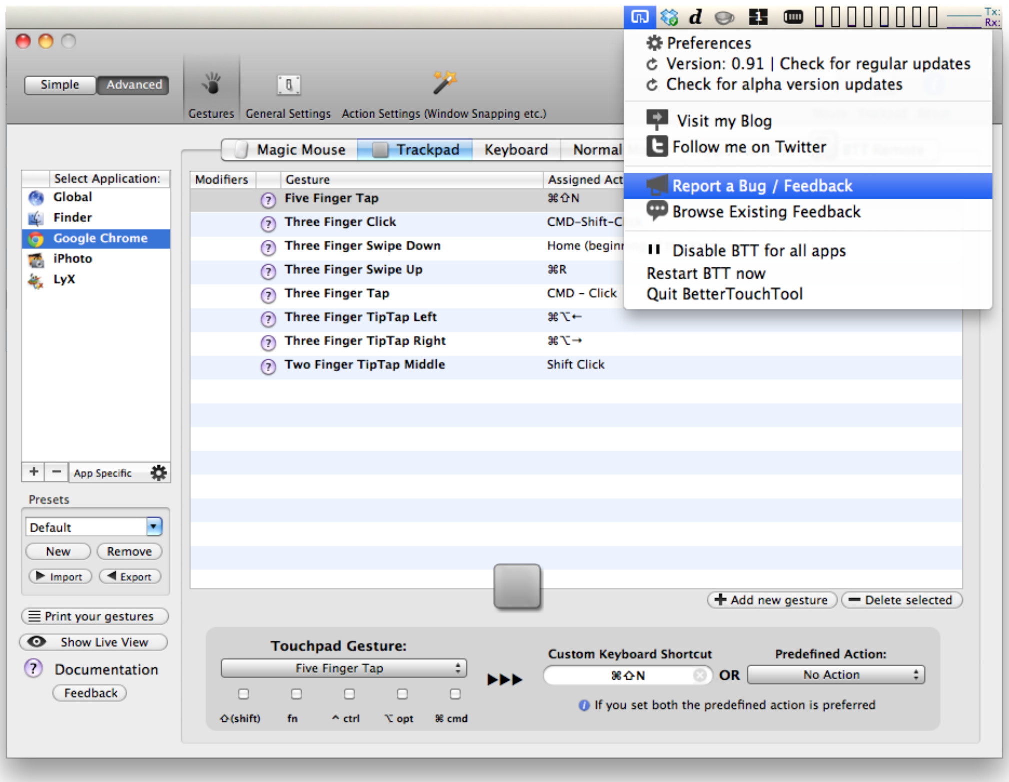
☑ I agree

Comment on this feedback:

☑ Provide a rating

Rate this feedback: 3

Figure 6.4: PORTNEUF for MOSKitt: Example of a user feedback recommendation and collective user feedback.



| Modifiers | Gesture | Assigned Act |
|---|---|---|
| | Five Finger Tap | ⌘⇧N |
| | Three Finger Click | CMD-Shift-C |
| | Three Finger Swipe Down | Home (beginn |
| | Three Finger Swipe Up | ⌘R |
| | Three Finger Tap | CMD - Click |
| | Three Finger TipTap Left | ⌘⌥← |
| | Three Finger TipTap Right | ⌘⌥→ |
| | Two Finger TipTap Middle | Shift Click |

Touchpad Gesture: Five Finger Tap

Custom Keyboard Shortcut: ⌘⇧N OR Predefined Action: No Action

If you set both the predefined action is preferred

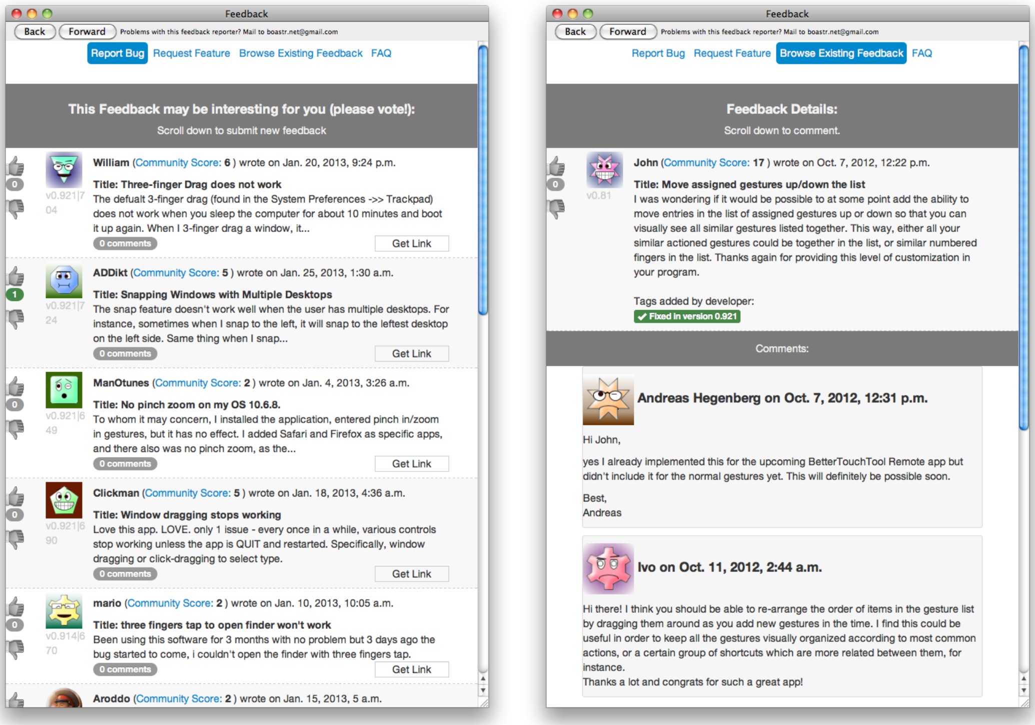
Figure 6.5: Integration of PORTNEUF in BetterTouchTool.

### 6.1.2 Summative Evaluation

The goal of the summative implementation of PORTNEUF was to gather collective feedback from a large user community, in order to measure and quantify real-world effects and to evaluate the generated recommendations. To this end, we developed a reusable, web-based version of the PORTNEUF framework using Python in combination with a MySQL database to hold user feedback and user experience snapshots. Further, we implemented a lightweight monitoring library including six generic sensors, capable of observing the context of arbitrary Mac OS X and iOS applications, as well as a user feedback library. Both components can be easily integrated into arbitrary Mac OS X and iOS applications.

We integrated PORTNEUF into BetterTouchTool[3], a desktop application for Mac OS X which allows users to configure trackpad, mouse, and keyboard gestures for arbitrary applications. BetterTouchTool has a large community with more than 200,000 active users. In order to obtain tailored recommendation results, two more application-specific sensors were added to the source code of

[3]http://www.boastr.de



Problems with this feedback reporter? Mail to boastr.net@gmail.com

**This Feedback may be interesting for you (please vote!):**
Scroll down to submit new feedback

0 v0.921|704
**William** (Community Score: **6** ) wrote on Jan. 20, 2013, 9:24 p.m.
**Title: Three-finger Drag does not work**
The defualt 3-finger drag (found in the System Preferences ->> Trackpad) does not work when you sleep the computer for about 10 minutes and boot it up again. When I 3-finger drag a window, it...

1 v0.921|724
**ADDikt** (Community Score: **5** ) wrote on Jan. 25, 2013, 1:30 a.m.
**Title: Snapping Windows with Multiple Desktops**
The snap feature doesn't work well when the user has multiple desktops. For instance, sometimes when I snap to the left, it will snap to the leftest desktop on the left side. Same thing when I snap...

0 v0.921|649
**ManOtunes** (Community Score: **2** ) wrote on Jan. 4, 2013, 3:26 a.m.
**Title: No pinch zoom on my OS 10.6.8.**
To whom it may concern, I installed the application, entered pinch in/zoom in gestures, but it has no effect. I added Safari and Firefox as specific apps, and there also was no pinch zoom, as the...

0 v0.921|690
**Clickman** (Community Score: **5** ) wrote on Jan. 18, 2013, 4:36 a.m.
**Title: Window dragging stops working**
Love this app. LOVE. only 1 issue - every once in a while, various controls stop working unless the app is QUIT and restarted. Specifically, window dragging or click-dragging to select type.

0 v0.914|670
**mario** (Community Score: **2** ) wrote on Jan. 10, 2013, 10:05 a.m.
**Title: three fingers tap to open finder won't work**
Been using this software for 3 months with no problem but 3 days ago the bug started to come, i couldn't open the finder with three fingers tap.

**Aroddo** (Community Score: **2** ) wrote on Jan. 15, 2013, 5 a.m.

**Feedback Details:**
Scroll down to comment.

0 v0.81
**John** (Community Score: **17** ) wrote on Oct. 7, 2012, 12:22 p.m.
**Title: Move assigned gestures up/down the list**
I was wondering if it would be possible to at some point add the ability to move entries in the list of assigned gestures up or down so that you can visually see all similar gestures listed together. This way, either all your similar actioned gestures could be together in the list, or similar numbered fingers in the list. Thanks again for providing this level of customization in your program.

Tags added by developer:
✓ Fixed in version 0.921

Comments:

**Andreas Hegenberg on Oct. 7, 2012, 12:31 p.m.**

Hi John,

yes I already implemented this for the upcoming BetterTouchTool Remote app but didn't include it for the normal gestures yet. This will definitely be possible soon.

Best,
Andreas

**Ivo on Oct. 11, 2012, 2:44 a.m.**

Hi there! I think you should be able to re-arrange the order of items in the gesture list by dragging them around as you add new gestures in the time. I find this could be useful in order to keep all the gestures visually organized according to most common actions, or a certain group of shortcuts which are more related between them, for instance.
Thanks a lot and congrats for such a great app!

Figure 6.6: PORTNEUF for BetterTouchTool: Example of a user feedback recommendation and collective user feedback.

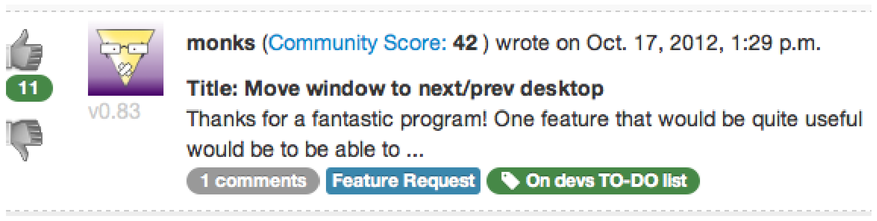
the application[4]. PORTNEUF is integrated into BetterTouchTool since version 0.799, allowing users to provide in situ feedback as depicted in Figure 6.5. The included PORTNEUF feedback library contains a simple user interface displaying the PORTNEUF web service results. Users of BetterTouchTool who want to provide feedback select the corresponding menu item. The user interface then shows up to 10 recommendations of existing user feedback generated by the PORTNEUF web service. PORTNEUF explicitly encourages users to vote for or comment on existing user feedback before creating a new entry. Figure 6.6 illustrates the generated recommendations and shows a selection of the collective user feedback.

End users are not the only group which may provide feedback in PORTNEUF. Developers can react to the user feedback, for instance by commenting it. Such back-feedback can be helpful to clarify misunderstandings or to ask for missing information. Additionally, a functionality which allows developers to assign visual tags to feedback was included. The goal of these tags is to allow developers to deliver a quick message to all end users who utilize PORTNEUF. We included 9 different tags and additionally assigned a reputation score to each tag, as

[4]The complete list of sensors is depicted in Table B.1.

Table 6.1: Developer feedback tags in PORTNEUF.

| # | tag | reputation points |
|---|---|---|
| 1 | working on this | 5 |
| 2 | on todo list | 4 |
| 3 | on todo list (low prio) | 2 |
| 4 | needs more votes | 0 |
| 5 | solution in comment | 0 |
| 6 | cannot reproduce | 0 |
| 7 | not a bug | 0 |
| 8 | won't change | 0 |
| 9 | not possible | 0 |


11

v0.83

monks (Community Score: 42 ) wrote on Oct. 17, 2012, 1:29 p.m.

Title: Move window to next/prev desktop

Thanks for a fantastic program! One feature that would be quite useful would be to be able to ...

Figure 6.7: PORTNEUF developer tag in BetterTouchTool end user feedback.

shown in Table 6.1. Whenever a developer assigns such a tag to a specific feedback, the end user who authored this feedback gains the specified amount of reputation points. The idea behind this mechanism is to create a strong feeling of involvement among the end users. We hypothesize that this bridges the gap between users and developers and represents an incentive for users, because they see that their voices are not ignored. Figure 6.7 depicts an end user feedback to which a developer tag was assigned. Moreover, we implemented a leaderboard which shows the twenty users with the highest reputation scores. Leaderboards are an essential method in the gamification movement, which is becoming more and more popular in novel applications (e.g. Foursquare[5]) [79].

## 6.2 Evaluation Setting

PORTNEUF should facilitate developers' work with user feedback by providing tool support for its consolidation and the assessment of its impact. More specifically, the goal of PORTNEUF is threefold. First, it should reduce the work overhead for developers created by unstructured feedback which typically contains numerous duplicates. Second, it should allow developers to assess as how important reported feedback is considered by the community. It should thus

[5]https://foursquare.com

provide a comprehensive, quantitative picture of the feedback impact, which facilitates its prioritization. Third, it should bridge gaps between users and developers which typically deteriorate the benefits of user involvement for both stakeholders.

The purpose of this summative evaluation is to test if PORTNEUF successfully reaches its goal, and to get insights about emerging issues as well as suggestions for future improvement. We first formulate the questions that this evaluation will answer. Then, we describe the methodology we used to collect and analyze the evaluation data. Finally, we present the actual evaluation data sets collected to perform our analysis.

### 6.2.1 Evaluation Questions

The evaluation questions refine our RQ 6 and investigate if PORTNEUF increases developers' efficiency when working with user feedback. Specifically, we study the effects of *user feedback recommendation* and *impact assessment*, and strive for collecting emerging *issues and improvement* suggestions.

**RQ 6.(a) User feedback recommendation** describes to which extent user feedback recommendations in PORTNEUF facilitate developers' work with user feedback. In particular, we investigate the following questions:

- *Duplicates*: Does PORTNEUF reduce the amount of duplicate user feedback?
- *Effects*: Which effects do recommendations have on the provided feedback?
- *Quality*: How relevant are the recommendations and which are their limitations?

**RQ 6.(b) Impact assessment** describes if PORTNEUF assesses user feedback impact in a way which is helpful for developers. In particular, we study the following questions:

- *Prioritization*: Do developers regard the provided prioritization of user feedback sensible and representative for the community?
- *Impact*: Do developers relate to the user feedback impact provided by PORTNEUF?

**RQ 6.(c) Issues and improvements** summarizes emerging issues and suggestions for future improvements. We investigate the following questions:

- *Satisfaction*: How satisfied are developers with PORTNEUF?
- *Issues*: Which issues were encountered and how could these be solved?

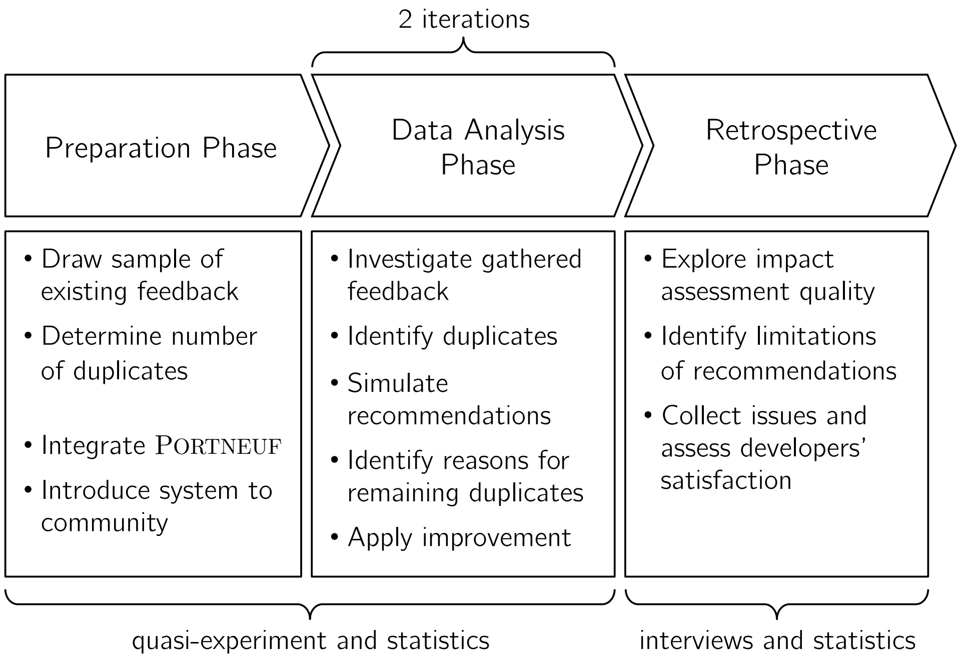

2 iterations
Preparation Phase
Data Analysis Phase
Retrospective Phase
• Draw sample of existing feedback
• Determine number of duplicates
• Integrate PORTNEUF
• Introduce system to community
• Investigate gathered feedback
• Identify duplicates
• Simulate recommendations
• Identify reasons for remaining duplicates
• Apply improvement
• Explore impact assessment quality
• Identify limitations of recommendations
• Collect issues and assess developers' satisfaction
quasi-experiment and statistics
interviews and statistics

Figure 6.8: Evaluation method.

In order to answer these questions, we evaluated PORTNEUF in the application BetterTouchTool. The evaluation took place between October and December 2012. During this time, we worked closely together with the developers of the application.

## 6.2.2 Evaluation Methodology

We employed two main methods to answer the evaluation questions, a *quasi-experiment* [53] to measure the effects of PORTNEUF on user feedback, and an *interview* to qualitatively assess the results as perceived by developers. We used *descriptive statistics* to obtain quantitative evaluation results and test their significance.

The goal of an *experiment* is to test which impact a specific treatment has on a particular phenomenon. *Controlled experiments* control for other factors which might influence the phenomenon by using a *randomized* design. To this end, subjects are assigned randomly to either the *experimental group* or the *control group*, the former of which receives the treatment, the latter of which does not. In contrast, *quasi-experiments* use a nonrandomized design, typically because random assignment is impossible [72]. In our case, we could not use a randomized design since this would have required splitting the user community. Since PORTNEUF relies on the user community phenomenon to consolidate feedback, this would have affected the dependent variables. Instead, we formed the control group by drawing a historical sample of the studied *population*, namely user feedback for BetterTouchTool. The experimental group was formed by applying our treatment, i.e. PORTNEUF, to the user community and collecting the emerging feedback. The principal *dependent variable* in our experiment is the rate of duplicates among user feedback, as this measures the effort required by developers to deal with user feedback.

Our evaluation methodology consisted of three phases: a preparation phase, a data analysis phase, and a retrospective phase, as depicted in Figure 6.8.

#### 6.2.2.1 Preparation Phase

The preparation phase served two purposes. First, to understand the status quo of user feedback in the control group, i.e. the investigated system without PORTNEUF. Second, to integrate PORTNEUF into the target software BetterTouchTool, i.e. to apply the treatment, to kick off the evaluation period.

We selected a sample of 2,000 successive reports from existing user feedback, which the company had received via email between July 2010 and September 2011. This feedback formed the control group. Because duplicate feedback was not systematically tagged, we asked a BetterTouchTool developer to manually work through the data set and mark duplicates. The obtained result served as benchmark for the further evaluation of PORTNEUF.

Next, we deployed the PORTNEUF web service, and integrated the monitoring and user feedback libraries into BetterTouchTool. After testing the system and feedback functionality, BOASTR released the PORTNEUF powered BetterTouchTool (version 0.799) on 6 October 2012.

#### 6.2.2.2 Data Analysis Phase

We split the data analysis phase in two iterative phases, lasting 4 weeks and 3 weeks, respectively. During both time frames, end users provided feedback about BetterTouchTool using the integrated PORTNEUF framework. After each phase, we collected the data from the experimental group in the form of a snapshot of the accumulated feedback, and analyzed the data. We used the insights gained after the first phase to apply small improvements to the PORTNEUF monitoring component, and then released a version containing the changes.

Each data analysis phase consisted of the following steps. First, in order to assess changes in the work overhead, we asked a BOASTR developer to manually identify duplicates, so we could compare the resulting duplicate rate, which represents the dependent variable in our experiment, to the amount of duplicates obtained without PORTNEUF. To understand the reason for the remaining duplicates, we then studied the recommendations which had been provided by PORTNEUF to the reporters of these duplicates, and manually analyzed the duplicated items. Last, we investigated the user involvement degree using de-

Table 6.2: Evaluation data sets.

| # | data set | time frame | #feedback | #users | fpu | fpd |
|---|---|---|---|---|---|---|
| $D_C$ | historical feedback (control group) | 14 months | 2,000 | 1,576 | 1.27 | 4.65 |
| $D_1$ | first evaluation period (part 1 of exp. group) | 4 weeks | 202 | 188 | 1.07 | 6.31 |
| $D_2$ | second evaluation period (part 2 of exp. group) | 3 weeks | 119 | 109 | 1.09 | 5.67 |
| $D_E$ | complete evaluation period (experimental group) | 7 weeks | 321 | 288 | 1.11 | 6.07 |

scriptive statistics to explore the obtained error reports, feature requests, votes, and comments in combination with the corresponding recommendations.

#### 6.2.2.3 Retrospective Phase

In the retrospective phase, we explored the quality of the impact assessment and feedback prioritization created by PORTNEUF. We further investigated remaining questions which could only be answered by the BOASTR developers. For instance, we wanted to determine the helpfulness of the bidirectional communication channel in PORTNEUF which allows developers to ask for clarifications directly and personally. Moreover, we identified cases where user feedback recommendations did not prevent duplicates in order to summarize limitations and their respective reasons. Finally, we collected issues which were raised by the developers during the evaluation period and interviewed them with respect to their overall satisfaction.

### 6.2.3 Evaluation Data

Table 6.2 shows an overview of the evaluation data, which consists of three data sets. Data set $D_C$ contains the control group, i.e. historical feedback, which spans a time frame of 16 months, and includes 2,000 feedback reports from 1,576 distinct users. This makes around 1.27 reports per user and about 4.65 reports per day.

Data set $D_1$ contains all feedback collected during the first evaluation period. It spans 4 weeks and comprises 202 reports from 188 distinct users, leading to on average 1.07 feedback per user and around 6.31 reports per day. After the first evaluation period, we improved the monitoring component, so that users worked with this new version during the second evaluation period. Data set $D_2$ contains all the feedback collected during the second evaluation period. It spans 3 weeks and consists of 119 reports from 109 distinct users, leading to on average 1.09 feedback per user and around 5.67 reports per day. Together, our experimental group $D_E$ spans 7 weeks and comprise 321 reports from 288 distinct users, leading to on average 1.11 feedback per user and around 6.07 reports per day.

## 6.3 Evaluation Results

We first investigate the results of user feedback recommendations in terms of created duplicates and interpret the occurrent effects. Next, we explore the quality of PORTNEUF's impact assessment algorithm. Last, we describe emerging effects from developers' point of view and summarize issues and possible improvements.

### 6.3.1 User Feedback Recommendation

Collective user feedback essentially depends on an effective way to bring together otherwise separated users and their experiences and opinions. Consequently, the main enabler for collective user feedback is generating relevant user feedback recommendations. The goal of PORTNEUF is to reduce the number of duplicate feedback using these recommendations.

#### Duplicates

To allow for a comparison, we first measured the amount of duplicates in the control group $D_C$. We found that 846 (42.3%) of the 2,000 reports we had sampled were duplicates of 274 original reports. This means, if a report was duplicated it led to around 3.09 additional reports. Next, we analyzed the feedback obtained in the two evaluation periods. We found that of the 202 reports in $D_1$ only 29 (14.35%) were duplicates. The resulting difference in the duplicate rate is statistically significant ($p < 0.001$). We counted 21 duplicated reports, leading to around 1.38 additional reports per duplicate. Data set $D_2$, which corresponds to the second evaluation period, contains 119 reports, of which only 17 (14.29%) were duplicates. Again, the resulting difference in the duplicate rate is statistically significant ($p < 0.001$). We counted 6 duplicated reports, leading to around 2.83 additional reports per duplicate.

In order to compensate for variation of the duplicate rate during the time period $T = [t_0, t_\omega] \subseteq \mathbb{R}_0^+$, we calculated the proportion $\Delta_T$ between the definite integrals of both duplicates and total feedback as follows. Let $\mathit{duplicates} : \mathbb{R}_0^+ \to \mathbb{N}_0^+$ and $\mathit{feedback} : \mathbb{R}_0^+ \to \mathbb{N}_0^+$. For a given time $t$, $\mathit{duplicates}(t)$ refers to the number of duplicates at time $t$ and $\mathit{feedback}(t)$ denotes the total amount of feedback at time $t$. Both functions are measurable. Thus, we may interpret $\Delta_T$ as the ratio between the areas under the graphs representing the amount of duplicates and total feedback during $T$. It is calculated as follows:

$$\Delta_T := \frac{\displaystyle\int_{t_0}^{t_\omega} \mathit{duplicates}(t)\,\mathrm{d}t}{\displaystyle\int_{t_0}^{t_\omega} \mathit{feedback}(t)\,\mathrm{d}t}\,, \tag{6.1}$$

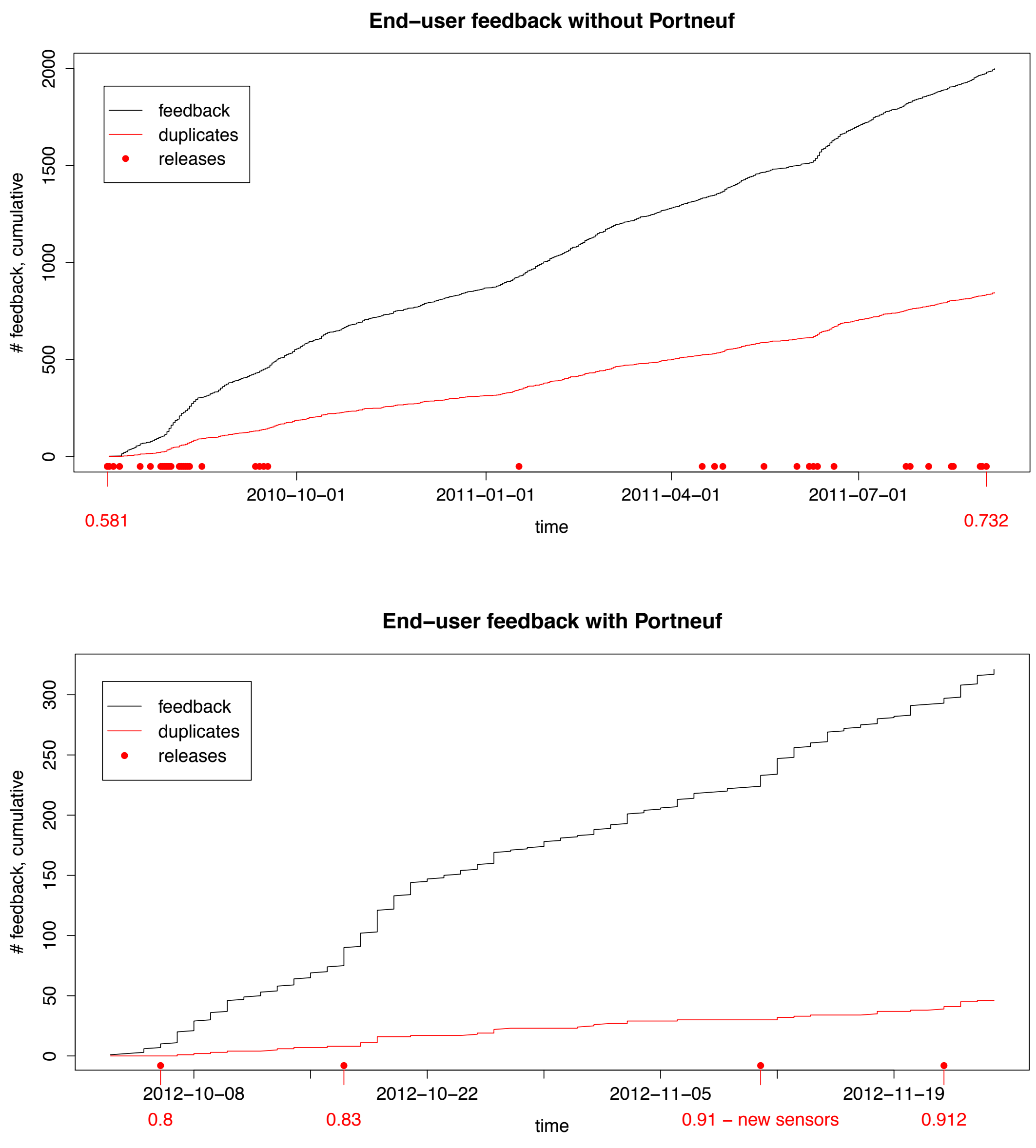


Figure 6.9: End user feedback characteristics without and with PORTNEUF. The black curve depicts the cumulative amount of feedback over time, i.e. *feedback*$(t)$. The red curve shows the cumulative number of duplicates over time, i.e. *duplicates*$(t)$. Dots illustrate releases of BetterTouchTool.

We obtained a value of $\Delta_{D_C} = 0.389$ for the control group $D_C$, and a value of $\Delta_{D_E} = 0.128$ for the experimental group $D_E$. This means that users in the experimental group created $\frac{\Delta_{D_E}}{\Delta_{D_C}} = 67.1\%$ less duplicates than were made in the control group.

Last, we averaged over the fraction of duplicates calculated for each data point in $D_C$ and $D_E$. For the control group $D_C$ we obtained a mean of $\overline{D_C} = 35.7\%$ duplicates per feedback. In contrast, we obtained a rate of only $\overline{D_E} = 11.2\%$ in the experimental group. A two-sample t-test and a two-sample Wilcoxon rank sum test give strong statistical evidence for the significant difference between the duplicates in the data sets $D_C$ and $D_E$ in terms of both means and medians ($p < 0.001$). Figure 6.9 illustrates the results.

To determine the effect size of duplicate reduction, we calculated Cohen's $d$ [68], which measures the strength of a phenomenon by indicating the standardized difference between two obtained means. A value of $d \leq 0.2$ represents a small effect, a value of $d = 0.5$ represents a medium effect, and a value of $d \geq 0.8$ represents a strong effect [68]. To calculate the effect size, we estimated the difference between the means of the duplicate rates in $D_C$ and $D_E$ and divided it by the *pooled standard deviation* $\sigma_p$ [214]:

$$d = \frac{\overline{D_C} - \overline{D_E}}{\sigma_p}, \text{ with } \sigma_p = \sqrt{\frac{(\|D_C\| - 1)\,\sigma_C^2 + (\|D_E\| - 1)\,\sigma_E^2}{\|D_C\| + \|D_E\|}} \tag{6.2}$$

We obtained a value of $d = 4.27$, which denotes a *strong* effect. Descriptively, this means that the average fraction of duplicates with PORTNEUF is 4.27 standard deviations below the average fraction of duplicates without our framework. Note that we utilize a pooled standard deviation to give consideration to our quasi-experimental setup, which does not guarantee that the populations from which $D_C$ and $D_E$ were drawn are identical.

## Effects

To understand the effects of recommendations on end users and the emerging user feedback, we investigated how users interacted with PORTNEUF after having obtained user feedback recommendations. To this end, we tracked users' interactions with PORTNEUF throughout the evaluation period. Figure 6.10 depicts the results and shows the frequency of each interaction path among all interactions.

We registered a total of 584 interactions during the evaluation period. During 321 (55.0%) interactions users eventually created new feedback. This feedback corresponds to our evaluation data set $D_E$. As shown in Figure 6.10, 46 (7.9%) interactions led to the creation of duplicates, while in 275 (47.1%) cases, users provided unique feedback. We found that in 9 (1.5%) interactions which led to

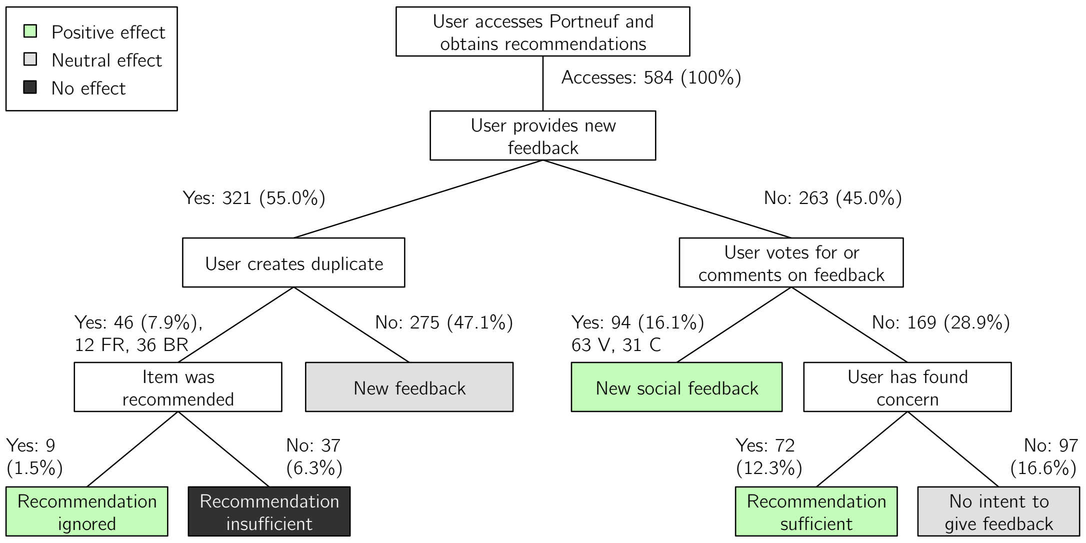


Figure 6.10: Effects of user feedback recommendations in PORTNEUF.

duplicates, users had simply ignored a correct recommendation. In contrast, in 37 (6.3%) interactions, users did not obtain a suitable recommendation. While the duplicate rate significantly decreased after introducing PORTNEUF, we conclude that the observed rate might still be improved.

We further observed that 263 (45.0%) interactions did not lead to the creation of new feedback. In the light of the reduced duplicate rate, we can argue without loss of generality that without PORTNEUF a non-empty subset of these interactions would have led to additional feedback and presumably more duplicates. For instance, to reach the duplicate ratio of the historical sample, more than 58% of these 263 interactions would have to provoke duplicates. In 94 (16.1%) cases users provided social feedback on existing reports, including 63 votes and 31 comments. We are confident to argue that most of these would have led to duplicate feedback without PORTNEUF.

The remaining 169 (28.9%) interactions ended without any feedback. To understand why the corresponding users had not provided feedback, we had included a short dialog which asked the user if she found her concern, given that she closes PORTNEUF without any further contribution. In 72 (12.3%) cases, users answered that their concern was already described. We conclude that these may be counted among the positive effects of user feedback recommendations. In the other 97 (16.6%) cases users indicated that their concern was not among the recommendations. Our interpretation of these cases is that the corresponding users did not have the intent to provide feedback. In any case, the recommendation did not have any effect during the corresponding interactions.

To summarize the effects of user feedback recommendations, we classify the corresponding interaction results into three kinds of effects. With *"positive ef-*

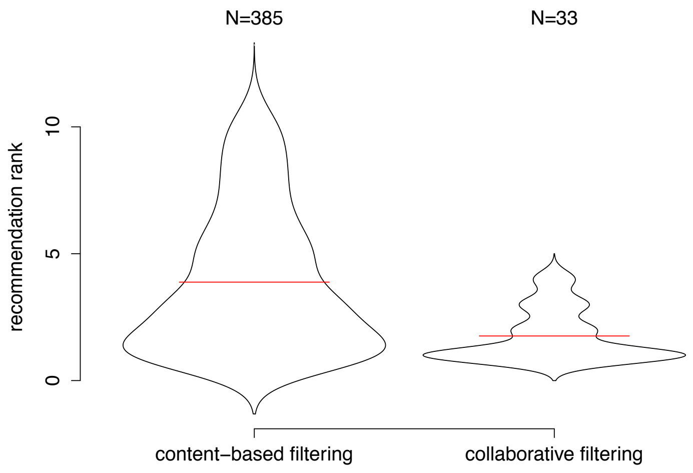


Figure 6.11: Recommendation ranks in PORTNEUF.

*fect"* we denote that recommendations included relevant feedback. With *"no effect"* we describe that relevant feedback was not recommended. Finally, with *"neutral effect"* we denote all cases which could not lead to effects even if perfect recommendations were given.

As illustrated in Figure 6.10, in 175 (30.0%) cases recommendations had a clearly positive effect. They helped to avoid duplicates and provoked the creation of social feedback, which facilitates impact assessment. In only 37 (6.3%) cases user feedback recommendation had no effect, meaning that users created duplicates in spite of reading the generated recommendations. Finally, in 372 (63.7%) cases recommendations had a neutral effect. These cases describe scenarios which cannot be improved by user feedback recommendations.

## Quality

In total, PORTNEUF generated 584 lists with recommended feedback during the evaluation period. In 211 (36.1%) cases, users interacted with at least one of the corresponding recommendations – by voting for (50 or 8.6%), commenting (30 or 5.1%), or at least clicking on (197 or 33.7%) a subset of the items to read their details. In total, users interacted with 418 recommended items, leading to around 1.98 interactions per access. Most of these useful recommendations (385 or 92.1%) were created using content-based filtering, while only a small amount (33 or 7.9%) using collaborative filtering. Figure 6.11 illustrates the distribution of the useful recommendations in terms of their ranks on the recommended items list. On average, collaborative filtering led to higher ranked recommendations

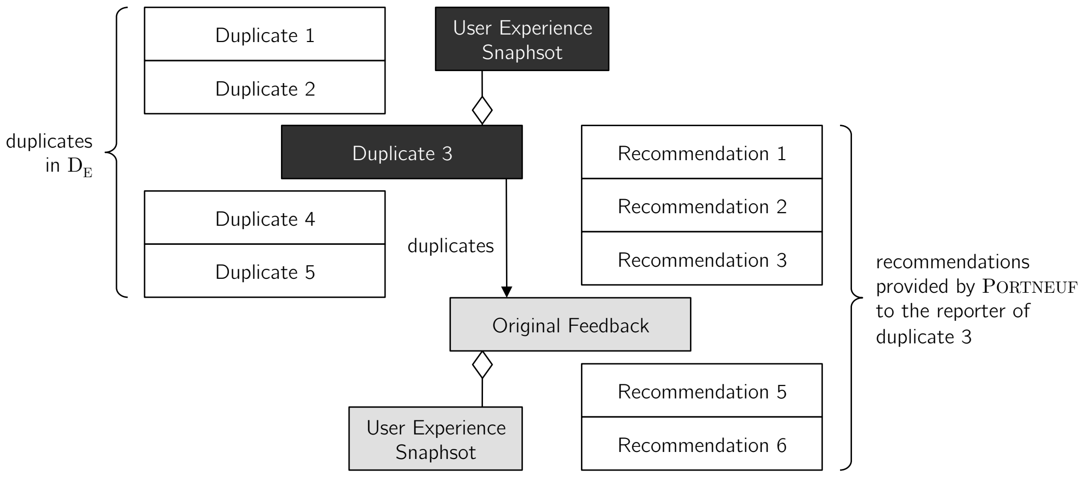


Figure 6.12: Analysis of ineffective PORTNEUF recommendations.

(mean rank 1.76) than content-based (mean rank 3.88), but it was utilized by far less often to generate recommendations.

*Mean average precision* [191] is frequently used in information retrieval to measure the quality of systems that return multiple results to a query. It provides a single value which represents the quality of the results based on relevance and rank in the result list. The mean average precision for the recommendations generated during the evaluation period equals 44.2%. However, the goal of PORTNEUF is to identify the *one* item, which the current user is about to report – if it exists – and recommend it to the user. Therefore, measuring usage prediction quality expressed by precision and recall is less meaningful in our case [239], since only a small set of recommended items is relevant to the user. Instead, we calculate the *hit-rate* (HR), a utility-based ranking measure proposed by Desphande and Karypis [78]. It measures how often a recommender system was able to recommend the right item among the total number of queries:

$$\textit{hit-rate (HR)} := \frac{\textit{Number of hits}}{n} \tag{6.3}$$

An HR value of 1.0 indicates that the right item was always recommended, whereas an HR value of 0.0 denotes that not a single right item was recommended. To calculate the hit-rate for PORTNEUF, we omitted the 372 neutral cases where new feedback was provided or users did not intent to provide feedback, since there was no right item to be recommended. In 175 of the remaining 212 cases, PORTNEUF created the right recommendations. Consequently, the hit-rate of PORTNEUF equals $\frac{175}{212} = 82.5\%$.

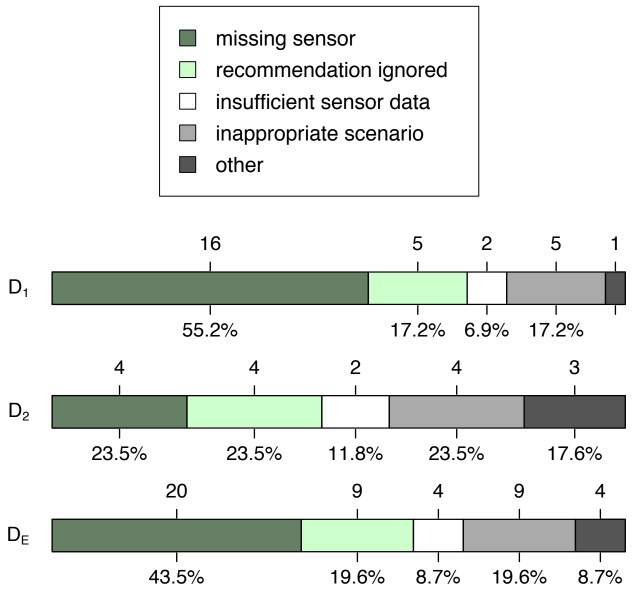


Figure 6.13: Reasons for duplicates in PORTNEUF.

To further investigate the quality of user feedback recommendation in PORTNEUF and identify its limitations, we analyzed the reason for each duplicate in the evaluation sets as follows.

First, we looked up the original report which had been duplicated. We then studied the recommendations which the user who created the duplicate had obtained, as illustrated in Figure 6.12. If the original report was among the recommended items, the user had ignored it – either because she did not read the recommendations or because she did not recognize her concern in the recommended items.

To identify the reasons for the remaining duplicates, we compared the user experience snapshots of duplicate and original feedback. The results are depicted in Figure 6.13. We found that after the first evaluation period a plurality of duplicates (55.2%) were due to missing sensors. Specifically this means that user experience snapshots did not include events which would have been necessary to allow the content-based recommendation strategy within PORTNEUF to create appropriate recommendations. For instance, one specific error report described that BetterTouchTool would always obtain the window focus when the system woke up from sleep. However, no sensor monitored the system sleep state, with the result that this error report ended up duplicated several times. After analyzing the duplicates from the first evaluation period, we identified two missing sensors and implemented them so that they could be used in the second evaluation period. As shown in Figure 6.13, the amount of duplicates which were due to missing sensors dropped to 23.5% during the second evaluation period.

Similarly, 8.7% of the duplicates were due to insufficient sensor data, mainly because of two reasons. First, when users provided feedback shortly after updating BetterTouchTool to the version containing new sensors, there was no time to collect enough relevant events. Second, if users reported problems not when they occurred but instead postponed their feedback, the corresponding events were outdated. Since user experience snapshots underlie a deliberate aging process, the importance of relevant events decreases during the further use of the application. Although user experience aging leads to problems in border cases, it is necessary to be able to identify relevant events in the current context.

We found that 19.6% of all duplicates were created because the particular scenarios were not appropriate for content-based user feedback recommendations. At the same time, we observed that collaborative filtering did not perform well due to the "ramp-up" problem [163]. Since collaborative filtering essentially compares users' preferences, users with few or no expressed preferences – i.e. votes or comments – are difficult to categorize. During the short time frame of PORTNEUF's evaluation, only 224 votes and 150 comments had been provided by 207 users in total. We observed around 1.91 votes and 1.19 comments per user – not enough to derive significant user preferences.

The remaining 8.7% of the duplicates were due to different reasons, including errors for which sensors would be difficult to implement and very special feature requests. For instance, a user reported that her settings were lost after each update. A different user requested a feature to disable mouse acceleration for specific applications.

### 6.3.2 Impact Assessment

To assess to which extent developers agree with the prioritization of user feedback estimated by PORTNEUF, we asked BOASTR to create a prioritized list of the most important user feedback in $D_E$. Their prioritization had to reflect the importance of user feedback among the user community. Then, we utilized the FEEDBACKRANK algorithm introduced in Section 5.3.5 to generate a prioritized list of user feedback utilizing the quantitative information stored in collective user feedback. Last, we compared both lists by calculating the average precision of FEEDBACKRANK.

We tried different values for the quantification of votes and comments, as well as for the normalization constant $\epsilon$ in the utilized REPUTATIONSCORE algorithm. The normalization constant corresponds to a decay factor, which determines how fast the influence of votes and comments decreases along paths in the transitive closure of social feedback. We obtained the best result when assigning a value of 2 for both votes and comments, and a value of $\epsilon = 0.1$ for the normalization constant.

Table 6.3 shows the results with a retrieval cutoff after the top 15 items. It illustrates that, as expected, votes are an important factor when assessing

Table 6.3: Evaluation of impact assessment.

| FeedbackRank | feedback title | dev.rank. | votes |
|---:|---|---:|---:|
| 6.46 | Four Finger Double Tap | 3 | 19 |
| 5.55 | Two Finger Tap on Trackpad? | 1 | 9 |
| 4.67 | Volume swipe! | 2 | 10 |
| 3.99 | Double Tap | 4 | 14 |
| 3.91 | bttfeedback:// links not working in Chrome | – | 0 |
| 3.89 | Hangs/crashes after sleep or switching users | 9 | 7 |
| 3.42 | 4 finger | – | 6 |
| 3.03 | Dictation mouse shortcut | 8 | 6 |
| 2.96 | Hide the menubar/taskbar icon | – | 3 |
| 2.91 | Fit windows to new screen size | 6 | 8 |
| 2.84 | 4-Finger Taps register twice | 11 | 6 |
| 2.83 | Move/resize windows w/ keyboard | – | 7 |
| 2.65 | Move window to next/prev desktop | 10 | 10 |
| 2.64 | Center Window - 1/4 screen | – | 7 |
| 2.60 | CPU Usage at 95% | 5 | 4 |

feedback impact among the user community. We calculated the *average precision* [191] of the results obtained by FeedbackRank by comparing them to the list assembled by Boastr. We obtained a value of 76.1%, which suggests that FeedbackRank provides a valuable estimation of what is important to the community from the developers' point of view.

To understand this perspective more clearly, we showed the result to a Boastr developer. She confirmed that the ranking results are sensible, but admitted that clearly distinguishing the importance of some recommended items is not straightforward. In fact, according to her, all feedback contained in Table 6.3 was important. She also hypothesized that FeedbackRank's helpfulness will increase over time and with the size of a software project, since more feedback and more features complicate the kind of estimation the algorithm supports.

## 6.3.3 Issues and Improvements

### 6.3.3.1 Satisfaction

After the data analysis phase, we interviewed a Boastr developer to understand if Portneuf met developers' demands. The developer assigned Portneuf the best grade on a 5-point Likert scale explaining that she was very satisfied with the system. Two effects were perceived particularly positive: reduced work overhead and more efficient involvement of the user community.

**Reduced Work Overhead** PORTNEUF reduced the overhead for working with user feedback in two ways. First, it introduced collective user feedback, which significantly reduced the amount of duplicates, provided for the consolidation of otherwise scattered feedback, and enabled the collection of the various opinions and volunteered resources of multiple end users. The BOASTR developer reported that end users regularly commented on existing feedback instead of providing new feedback from scratch. She explained that in case of bug reports, this often led to additional information which helped to solve the issue:

> "We recently encountered a bug where the software would simply crash, which was reported by a user running an outdated OS version. But when we replied that we think it would be due to this old version, other users commented that they had the same problem despite running the current version. In the past, combining these different reports would have been much more difficult. Now, we had one thread with the problem described at the top and different perspectives by the users below."

Second, PORTNEUF reduced the amount of effort needed to assess the impact of user feedback. On the one hand, collective user feedback by itself accounts for this improvement, since it contains otherwise scattered items. The developer stated that users voted for features of which the company would have never thought: "If these users would have written mails distributed over a longer time, as in the past, I cannot imagine that this feature would have been implemented. We would have needed much more efforts in order to assess these kinds of requests." On the other hand, PORTNEUF's FEEDBACKRANK allows for an on demand prioritization and assessment of the most important feedback according to the user community, which can be helpful as starting point for work planning meetings and roadmap decisions.

**Increased Involvement Efficiency** When asked if it was helpful to be able to turn to specific end users in order to ask for clarifications, the developer agreed but stated that this was also possible before. She underlined that the really important improvement lies in the efficiency of the communication with PORTNEUF:

> "It was often the case that end users do not answer any more, if you reply to their feedback per mail. With PORTNEUF, even a different user might answer, for instance to prove if the issue has disappeared. So, the real advantage is that we can selectively communicate with the part of the community which is interested in or affected by a specific issue rather than only with a single user or the whole audience."

The developer was confident that end users will feel more involved with PORTNEUF: "I am sure that the users now have the impression to be closer to the developers. For instance, since they always have the same image and username, and also because they get scores for what they do." Last, she pointed out that Portneuf enables a more efficient and clearer overview of the feedback by the whole user community.

#### 6.3.3.2 Issues

During our evaluation, we collected the following suggestions for further improvement of Portneuf.

**Remaining Duplicates** Although Portneuf is able to significantly reduce duplicate user feedback, there is still room for improvement. We collected four main open issues whose resolution could help to further avoid duplicates. Three are of technical nature and regard user feedback recommendation, while the fourth is non-technical.

First, additional sensors could further improve the content-based recommender system. In theory, sensors should evolve as the system does, since they need to capture new kinds of user experience. In practice, the plurality of sensors could be generic, but the types of events must be dynamically extendable. Portneuf guarantees this extensibility by providing a public event interface, which can be specialized by sensor developers. After the first evaluation period, we had identified two important events which were not yet monitored by Portneuf, because we had compared user experience snapshots to identify the reasons for specific duplicates. Accordingly, we implemented the missing sensors and deployed a new version of Portneuf. However, the effect of these new sensors only appeared after the next duplicate, because the original feedback did not contain the essential events and was therefore not considered similar by the recommender system.

Second, as expected, feature requests can not be simply deduced by comparing past events. Therefore, content-based recommendation is not suitable to avoid duplicate feature requests. We did not collect enough user preferences during the evaluation to be able to generate recommendations based on collaborative filtering. Although the evaluation period did only last for 7 weeks, this issue could also be present after a longer time. Motivating users to provide social feedback is therefore important.

Third, we observed that user experience aging affects the recommendation performance. The effect is positive, as the events' age is one of the determining characteristics for their relevance. However, snapshots might get outdated when users do not report their concerns promptly. Portneuf is designed to support in situ feedback, which includes reporting where and when the concerns occurs. However, further research could be necessary to develop triggers which might identify problematic situations automatically and suggest to provide feedback at a convenient time.

Last, we found that the user feedback title is crucial. In several cases users did not recognize their concern in the titles they read while browsing the recommended items. The reason was almost always that these titles were unclear or too unspecific. When asking BOASTR how to deal with such cases, they suggested that developers should be able to edit user feedback titles, in order to clarify them.

**Performance and Scalability** During the 7 weeks evaluation period, PORTNEUF accumulated over 200,000 events in over 10,000 user experience snapshots. Although BetterTouchTool has a large user audience ($>$ 200,000), performance and scalability are definitely issues for the applicability, since there are even larger communities and usage should span longer periods. Currently, it takes about 4.8 seconds before PORTNEUF provides a recommendation. This includes the time to upload a snapshot over the internet, to compare it with the existing snapshots, and obtain the most relevant candidates. While this time is still acceptable [205], it might increase during further use, since more existing snapshots have to be analyzed. Therefore, it is worthwhile to investigate heuristics and special indexes which might reduce the time needed for the comparison, or to discard older snapshots to save calculation time and space.

**Collaborative Filtering and End-User Motivation** As discussed above, collaborative filtering depends mainly on expressed user preferences – i.e. social feedback such as votes and comments. During our evaluation, PORTNEUF was not able to accumulate enough social feedback to make reliable predictions about the preferences of users who requested new features. The underlying issue is that users are not motivated enough to provide social feedback. Currently, their only incentive is – among helping the developers – the list of the most important users for the community, where their name might appear. There is still room for increasing the incentive to contribute, for instance by introducing real benefits or further gamification elements.

**Missing Developer Dashboard** During the retrospective interview with the BOASTR developers, we found that one important improvement for PORTNEUF is a comprehensive developer dashboard. To further support working with user feedback, developers need an administrative tool which visualizes the collected feedback and its impact, and facilitates daily tasks such as reviewing new feedback, assigning tags, and communicating with end users. Further, it should be possible to connect PORTNEUF with internally used issue trackers, to allow for an end-to-end software evolution workflow. Last, we found that user experience snapshots contain valuable diagnostic information which can be very helpful to understand what users did before errors occurred and eventually what led to the corresponding errors. However, currently there is no straightforward way to analyze these snapshots and the enclosed events rather than reading and comparing them manually. We claim that visualizing this information and supporting comparisons, for instance by "user experience snapshot diffs" can help to identify missing sensors, to understand errors, and to fix error causes.

## 6.4 Summary

In this chapter we described the evaluation of the PORTNEUF framework. We tested two main concepts, proactive and context-aware recommendation of user feedback based on users' experience, as well as impact assessment based on collective user feedback. Our findings can be summarized as follows:

- We implemented the PORTNEUF framework for two different applications and successfully developed sensors capable of monitoring numerous user interaction and application execution events. The captured user experience could successfully be utilized to generate relevant user feedback recommendations. This shows the *feasibility* of our concepts.
- PORTNEUF increases developers' efficiency when working with user feedback in three ways.
  - It significantly reduces the amount of *duplicates* among user feedback by over 67%, which is a strong effect. The main enabler of our approach, content-based recommendations based on monitored user experience, has a hit-rate of 82.5%.
  - PORTNEUF allows for a more efficient *impact assessment*. On the one hand, collective user feedback collects otherwise scattered items and allows developers to quickly overview the whole community in a structured way. On the other hand, FEEDBACKRANK allows for an on demand prioritization of the most impactful feedback in the user community, and provides a starting point for work planning meetings and roadmap decisions.
  - The framework enables a more efficient *user* and *community involvement*. In particular it allows developers to selectively and bidirectionally communicate with the part of the community which is interested in or affected by a specific issue.
- BOASTR decided to keep using PORTNEUF as their primary user involvement solution, as they are very satisfied with the results. In cooperation with them we have collected emerging suggestions for improvement, such as how to further reduce duplicates, increase performance and scalability, and provide a dashboard for developers.

The evaluation of PORTNEUF is positive, and the results are promising. Developers benefit from PORTNEUF because it allows them to work more efficiently with user feedback. Also users benefit from the framework as their voices are more likely to be heard.

# Chapter 7
# Conclusions and Future Work

«*There is no reason anyone would want a computer in their home.*»

— Kenneth Olsen, founder of *Digital Equipment Corporation* (1977)

The goal of user involvement is to improve the usefulness of a system by understanding the needs and expectations of users. Like software development itself, user involvement should not end with the delivery of the system, but accompany and safeguard its evolution over time.

However, both hardware and software systems underwent fundamental changes over the last decades, and as a result also the actual users changed from engineers with special scientific requirements to "normal" people with manifold needs. Moreover, software systems have become part of our everyday life, and the mobile and ubiquitous devices we interact with have added an additional layer of diversity to the various application scenarios that software engineering and evolution has to deal with.

At the same time, it was never as important as now for software companies to meet their end users' expectations. On the one hand, free and open source competitors lurk at every corner, finding more and more supporters and gaining industrial relevance. On the other hand, end users themselves are getting more knowledgeable and exigent regarding the tools they want to use, and the quality they ask for. Todays' users grew up with high-speed internet, highly reactive, touch-based, natural user interfaces, context-aware software, and sensor-equipped, ubiquitous devices.

The questions is now, how can we as software engineers deal with these demanding people, who eventually pay for the systems we develop and for their applicability in an ever-changing environment? How can we give consideration to this new attitude towards our users?

The first step in this direction has been made by Boehm [41], who equipped us with an iterative development process that allows for continuous improvement which is more than necessary in current software engineering projects. Agile development processes like Scrum [261] and XP [21] have provided for the next step by embracing change and narrowing the gap between developers and users. We claim that now it is the turn of user involvement to make a next step – towards a systematic and continuous flow of information between users and developers.

In Chapter 3, we found that developers appreciate post-deployment user feedback as it helps to improve software quality and to identify missing features. But at the same time giving consideration to their opinions and expectations is difficult – mainly because the associated tasks require high reading, comprehension, and structuring efforts, in particular due to the high quantity and low quality of user feedback and since feedback is scattered across several communication channels.

Our approach to solve these issues is PORTNEUF – a framework that proactively consolidates user reports into collective user feedback by recommending relevant existing items based on a comparison of users' experience. By implementing PORTNEUF and evaluating it with a commercial application and a large number of real users, we could show that it has a real impact.

In this chapter, we draw conclusions from our work and emphasize questions which we consider as worthwhile to be researched in future. We summarize the contributions of this dissertation and pinpoint its implications (Section 7.1) and limitations (Section 7.2). Finally, we illustrate a research agenda covering topics and questions which emerge from our work (Section 7.3).

## 7.1 Contributions

With this dissertation we made three major contributions which enhance the body of knowledge on user involvement and tackle the identified problems. First, we provide empirical evidence on the status quo and desired state of user involvement during software evolution. Second, we describe a solution framework which we derive from a grounded theory on continuous user involvement. Third, we provide a reference implementation and detailed evaluation of the proposed framework.

### 7.1.1 Status Quo Assessment

We described the status quo of user involvement in research (RQ 1) and in software evolution practice (RQ 2). Our contributions can be summarized as follows:

1. During our literature review in Chapter 2 we found that user involvement is an established research field which includes various methods. However, due to the fundamental progress of hardware and software systems, the

distance between developers and users is steadily increasing, while at the same time users are becoming more demanding, knowledgeable, and extroverted. Application distribution platforms and mobile devices make it nearly impossible to know the actual users or the use context of software before actually delivering it. Yet there is only little research on how post-deployment user feedback influences software development and which benefits and challenges it provides. We also found that cost and benefit of user involvement determine its applicability in practice, since resource constraints often do not allow companies to employ effective user involvement methods.

2. In Chapter 3 we reported on an empirical case study [218] which we conducted with software professionals, in order to analyze the current practice of user involvement during software evolution, specifically for software with large user audiences. Our contribution spans the following three aspects:

    a) We identified a *need* for user feedback. We found that developers appreciate user feedback since it is helpful to improve software quality, to identify missing features, and to gather real-world usage data. On the other hand, users frequently provide feedback over different feedback channels. Their public feedback may apply pressure on software companies.

    b) We identified serious *problems* in current user involvement practice. We found that developers need to analyze what users write, in order to be able to react on user feedback. However, feedback is typically written in natural language, might have poor quality, and can include contradictions. Further, developers typically need to estimate feedback impact manually and consequently spend high effort on this assessment. As a result of these problems, developers often ignore feedback, simply sticking with their product roadmaps and development plans.

    c) We collected main *requirements* for user involvement tool support. In particular when feedback volume is high, developers need tools to consolidate, structure, analyze, and track user feedback. Grouping and counting similar or duplicate feedback and assessing feedback impact are especially important features.

### 7.1.2 Continuous User Involvement Framework

We established the theoretical foundations of our solution by deriving a grounded theory [111] on continuous user involvement which relies on user communities. To this end, we studied two phenomena in Chapter 4. On the one hand, we explored regularities in the communication between users and developers in open source communities [221] in order to understand how these two groups should be interweaved (RQ 3). On the other hand, we investigated how users currently provide feedback in application distribution platforms in order to understand if and how user feedback can be consolidated (RQ 4). Our contributions can be summarized as follows:

1. We showed the *feasibility* of user involvement relying on user communities. We found that users are willing and able to provide helpful feedback to developers when they are convinced that it improves the software they use. In open source communities boundaries between users and developers are low, and social media which allow both users and developers equally to express their opinions, seem to help further dissolving such boundaries. We found that users and developers frequently discuss about requirements, implementation, and community aspects. Developers on the one hand report about their recent development activities to communicate their project work to a broad user audience. Users on the other hand have their communication peak time shortly after new versions are released and report about their experiences with the software.

2. We identified regularities in user feedback which allow for its *consolidation*. We found that commercially relevant user feedback channels, such as application distribution platforms only allow for transactional communication. Similar as in open source communities, users frequently provide feedback particularly shortly after new releases, but this feedback is shorter and its quality lower. Users often include maintenance relevant shortcomings in their feedback and request improvements and additional features. Specifically, they share their experiences with the application, but typically only in positive feedback. We concluded that this reduces the likeliness that developers will be able to improve applications from such feedback alone.

3. We identified four *key factors* for successful continuous user involvement during software evolution, namely the personal involvement of users, bidirectional communication channels, the promotion of user communities, and the automated capturing of user experience.

We then substantiated these theoretical solution concepts into PORTNEUF – a context-aware and domain-independent model for the consolidation and prioritization of user feedback which harnesses the user community to group user feedback in a proactive way, while increasing its overall quality (RQ 5, Chapter 5). Our contributions can be summarized as follows:

1. We defined the context-aware and domain-independent PORTNEUF *model* which avoids the creation of unstructured, duplicate feedback by proactively recommending existing relevant feedback to users, which they can

rate, vote for, or comment on instead of creating feedback themselves. We formalized and conceptualized user experience – the enabling concept of this approach – and showed how it can be indirectly measured by monitoring events during application use. We further defined collective user feedback, which groups feedback of multiple users by their common experience. Because it captures the opinion of the user community as a whole, this enables a quantification of feedback impact.

2. We showed the *applicability* of the PORTNEUF model to three phases of a typical software engineering lifecycle, namely early design, system testing, and software evolution.

3. We provided a *reference software framework* which implements the PORTNEUF model for software evolution. We illustrated how to automatically capture users' experience and described an algorithm that creates feedback recommendations by calculating the similarity of this experience. Finally, we introduced the FEEDBACKRANK algorithm, which allows developers to assess the impact of collective user feedback in a quantitative way.

### 7.1.3 Implementation and Evaluation

We evaluated the PORTNEUF approach and framework in Chapter 6 by testing its two main concepts (RQ 6), proactive and context-aware recommendation of user feedback based on users' experience as well as impact assessment based on collective user feedback. Our contributions can be summarized as follows:

1. We showed the *feasibility* of our approach by providing a reference implementation of the PORTNEUF framework for two different applications. We were able to develop sensors to monitor various user interaction and application execution events. The captured user experience could successfully be utilized to create relevant user feedback recommendations.

2. We provided empirical *evidence* that PORTNEUF increases developers' efficiency when working with user feedback. We showed that PORTNEUF significantly reduces the amount of duplicate user feedback by over two thirds and that the main enabler of our approach, user feedback recommendations based on user experience is working as expected. Moreover, we could show that collective user feedback and FEEDBACKRANK allows for a more efficient impact assessment of user feedback.

Our evaluation shows that developers benefit from PORTNEUF since it allows them to work more efficiently with user feedback, while users benefit because their voices are more likely to be heard.

## 7.2 Limitations

Our approach has three limitations, which result from our focus on observable events, specific user involvement cultures, and incremental changes.

**Unobservable Events** PORTNEUF derives user experience from events observed while a user interacts with an application. A prerequisite of this approach is that events which determine user experience are *observable*. For the purpose of this dissertation we concentrated on interactive software. Part of users' experience with interactive software is by definition observable, as it includes the interactions carried out by the users on the user interface [140]. The corresponding response in the dialogue [55], the application's execution, can be assumed to be observable without loss of generality, since in theory it is always possible to create events directly in the source code.

In practice, however, there are events which are not or not yet observable. They range from obvious limitations like the user's state of mind to more realistic and relevant constraints such as sandboxing [114] which does not allow software like sensors to access applications outside of their runtime environment. However, the question to raise is what do we *need* to observe in order to be able to *compare* user experience? Unfortunately this question cannot be answered in a general way, as it strongly depends on the application at hand.

One of the strengths of the PORTNEUF framework is the generality of the monitoring approach and the generic design of user experience. As a result, PORTNEUF is application and technology independent. With this design we provide for adaptability, extensibility, and portability, but within the scope of practically observable events.

**No User Involvement Culture** This work provides a practical solution to problems that occur if developers need to work with a high volume of end user feedback. But not all companies suffer these problems. We can think of two main reasons.

First, there might be no or not enough feedback. There are companies which are not interested in user feedback as a driving force of improvement, or at least they don't gather feedback actively. A prominent example is Apple. Apple's former CEO Steve Jobs pinpointed their position in an interview 1989: "You can't just ask customers what they want and then try to give that to them. By the time you get it built, they'll want something new." [150] Similarly, there are application domains where end user feedback is not applicable, such as aircraft design or nuclear power plants. However, we claim that PORTNEUF might still be of benefit in such cases. As we have demonstrated in Chapter 5, the *user* role is by no means limited to that of an application end user. During early design phases, usability tests might be necessary to finalize product design. Shortly before a major release, a software product needs to be extensively tested in order to avoid shipping buggy software. Independently of the concrete scenario, PORTNEUF provides means to *consolidate and structure large amounts of feedback on executable software*, be it customers' feedback on early executable prototypes or system testers' feedback on a potentially shippable product increment [261]. To increase incentives for users to provide feedback remains as challenging task for future research.

Second, companies might not want user feedback to be public or don't want to bias their users with other users' feedback. In our empirical study of user involvement practice described in Chapter 3, we found that there are companies which interact with their users in an almost cooperative way. In particular, if end users themselves are software professionals this might be the case more frequently. We have presented PORTNEUF as online platform, which allows users to vote for, rate, and comment on existing feedback of other users. But there might be cases where such a platform infringes users' privacy or restricts their creativity. We claim that PORTNEUF can still provide valuable benefits in such scenarios. On the one hand, monitored events – when symbolized – can lead to additional insights on how to reproduce and how to solve errors. On the other hand, PORTNEUF can also be used reactively and internally: Users can provide feedback which is enriched by user experience snapshots without interacting with other feedback. At any given time, *developers* can then use PORTNEUF on the accumulated set of feedback. The framework goes through this feedback one by one and repeatedly generates a list of recommendations based on the previously analyzed feedback, similar to a simulation. As a consequence, a developer only needs to read the recommended items and mark duplicates if she identifies them. Thus, in this scenario PORTNEUF becomes a feedback triage support system and will still reduce developers' required efforts.

**Fundamental Changes** PORTNEUF supports a critique based approach to software improvement, which contains three steps: *deployment*, *use*, and *feedback*. Consequently, PORTNEUF requires executable software, which users can obtain, use, and then provide feedback on. Such an approach is more suitable to discover incremental improvements and single missing features (cf. IKIWISI, [39]), than to call for paradigm breaks and revolutionary innovation [65, 282]. On the other hand, whenever the software company itself decides for a paradigm break, user feedback volume will experience a peak. In this case, PORTNEUF is the suitable tool to handle the emerging feedback.

## 7.3 Future Work

In the following, we illustrate a research agenda covering questions which emerge from our work and applications of our results which were out of the scope of this dissertation. We start by summarizing open issues in continuous user involvement which might influence the practical adoption of our approach. Then we describe how our results can support software engineers' decision making by externalizing the user community's preferences. Finally, we illustrate applications of user experience in other domains.

### 7.3.1 Open Issues in Continuous User Involvement

Continuous user involvement is still in an early stage [217]. As a consequence, there are several open questions and issues which future research should tackle. We summarize two major issues we have found during the empirical evaluation of our approach and suggest directions for appropriate solutions.

#### 7.3.1.1 Privacy

PORTNEUF monitors users' interactions with an application and builds user profiles from the observed events. In typical scenarios with end users the included information is sent over the internet to the development team. Once such information leaves the end users' site, privacy is a major issue. The monitored data might leak sensitive information such as passwords, social security numbers, file names, or the content of files. In order to protect users' privacy during continuous user involvement, future research should develop mechanisms to anonymize the monitored data without loosing information necessary to provide recommendations.

For instance, error reproduction research has shown that it is often possible to replicate a specific application execution even when any user input is altered [58, 67]. The proposed mechanisms rely on the fact that it is sufficient to select fake user input which triggers the same execution path as the real user input. For instance, an conditional statement providing for youth protection might check if the user's age is above 21. In this case, any value greater than or equal 21 will trigger the execution of a specific branch. As a consequence, there is no need to reveal the real age of the user. Altering the entered information to the corresponding branch condition, still triggers the same application execution behavior, but it protects the user's privacy.

Future work should investigate how such mechanisms influence the information quality in the more general case of user feedback. A specific question of interest is, which influence anonymization has on the recommendation quality and duplicate rate.

#### 7.3.1.2 Motivation and Incentives

In particular the recommendation of feature requests depends on available user preferences, in the form of social feedback. During our evaluation we found that under-contribution is a real issue due to the ramp-up problem [163] of collaborative filtering. The underlying issue is that users do not directly benefit from assigning votes or ratings to existing feedback. As this also takes time, users need to be motivated to contribute. Future research therefore should explore, which incentives can be offered to provoke more contributions.

Beenen et al. [22] have shown that social psychology predicts which incentives need to be given to motivate users to contribute to online communities. A different promising direction is *gamification*, the "use of game design elements in non-game contexts" [79]. Research in other areas has shown that gamification can motivate users and even influence their behavior. For instance, Singer and Schneider [264] successfully encouraged developers to frequently commit their working copies. Centola [59] illustrated how social reinforcement can spread specific behavior in a social network. In fact, we already included two gamification elements in the framework implementation we used during our evaluation: a leaderboard where the community could find the most influential users, and tags (or "badges") assigned by developers to particularly helpful feedback or feedback which made it on the development backlog. However, further research needs to be done to understand and optimize the occurrent effects.

But apart from the quantity of user feedback, users' motivation also determines its quality. Zimmermann et al. [298] propose CUEZILLA, an issue tracker add-on which should help reporters to submit bug reports with a higher quality. The authors included several incentives for reporters, including a quality meter indicating the current quality of the report and tips why specific entities are important to include. Future research should transfer the existing results to general user feedback and evaluate the emerging effects.

## 7.3.2 Social Software Engineering Decisions

We argue that PORTNEUF is inherently a decision support system, as it externalizes otherwise hidden information about the user community which might influence decisions in a software project. Engineering software in a social way includes considering users as first order citizens during decision making [182]. We describe three cases of how software engineering decisions can be supported by informing developers and managers of users' preferences and suggest directions for corresponding future work.

### 7.3.2.1 User Satisfaction Dashboards and Metrics

For the purpose of this dissertation, we focused our work on the feasibility and conceptual design of user feedback recommendations. Specifically, we did not work on one important aspect of the user involvement workflow: the connection of user feedback with the conventional development infrastructure. To be of more practical relevance to developers, the gathered collective user feedback

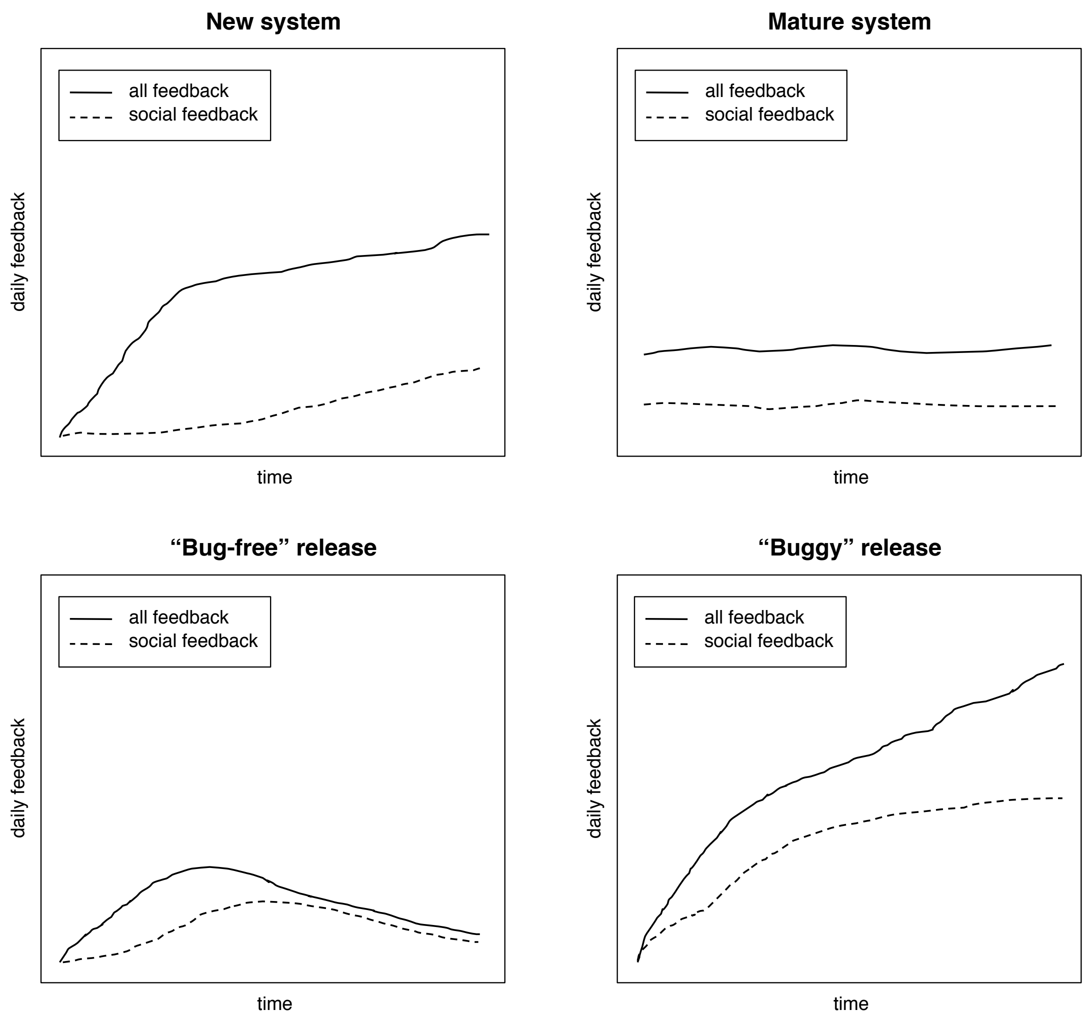


Figure 7.1: Possible feedback signatures for different system states.

and its characteristics need to be visualized in specialized dashboards [182]. Developers need to be able to assess emerging trends at a glance, to generate tables and figures for work planning meetings and reporting, and to link gathered feedback with other tools, such as issue trackers or CRM applications. Consequently, future research should explore developers' requirements on user feedback dashboards and investigate information needs and visualization requirements. Existing services such as Google Analytics[1] which analyze and visualize the user community of web pages, might be a starting point for this work. Their analysis yields statistics about the usage of a web page, grouped by characteristics of its users. This allows developers to draw conclusions about lost opportunities and success criteria based on when and where most users leave their web site.

[1]http://www.google.com/analytics/index.html

In addition, future work should investigate which conclusions can be drawn and which predictions made from the resulting trends. During our studies on user involvement in open source communities (Section 4.4) and application distribution platforms (Section 4.5), as well as in our evaluation (Section 6.3), we found evidence of a relation between release time and user feedback rate.

We therefore assume that the daily amount of new feedback and the corresponding fraction of social feedback exhibit a characteristic signature in different software lifecycle phases. The corresponding figures might allow for an assessment of user satisfaction and software quality. Figure 7.1 illustrates four possible signatures of user feedback. In the early stages of a new system, the total amount of feedback will continuously increase, followed by a similar increase of the social feedback fraction. In contrast, a mature system will show a rather uniform distribution of feedback and social feedback over time. In addition, we expect that the quality of a new release leaves its mark on the feedback signature. As illustrated, a bug-free release will trigger new feedback in the first couple of days after the release, but then feedback should return to a steady state. The corresponding fraction of social feedback will probably increase in a similar way but with a short delay. In contrast, if a release is rather buggy, we expect a steeper increase of feedback and a smaller fraction of social feedback.

We argue that PORTNEUF provides a starting point towards a set of measures which can support product and release managers' decision making. However, further research needs to be done to develop reliable and precise metrics from these measures, supposedly by collecting and studying real-world data in a hypothesis driven way.

#### 7.3.2.2 User Preference Conflicts

In software engineering, requirements verbalize decision alternatives regarding functionality and quality of the developed software [99]. Requirements for larger systems are typically negotiated by stakeholders with possibly contradicting preferences or even limited system and application domain knowledge. Not least due to these socio-technical issues, researchers consider requirements negotiation as one of the most critical and inefficient activities in software engineering projects [232]. Especially in projects where a large number of stakeholders is involved, conflicting preferences are frequent [99, 217]. To make matters worse, research has shown that stakeholders' preferences do not remain stable over time [98].

Users are particularly important stakeholders in software projects. In fact, the main goal of user involvement is to improve the usefulness of a system by understanding the needs and expectations of its users. But as we have found during our studies (cf. Section 3.3), also user preferences are often contradictory. PORTNEUF supports the continuous involvement of a large number of users by consolidating user feedback and facilitating the assessment of its impact.

However, future research needs to investigate how to *identify* and possibly *resolve* contradictions in the resulting collective user feedback.

The possibility to quantify social feedback could be a starting point for this work. Votings and ratings are by definition quantifiable, while comments might need a quantifiable extension which determines the comment type or included sentiment. For instance, Schneider et al. [260] present ConTexter, a framework which facilitates gathering feedback in context with mobile devices. When providing feedback with ConTexter, reporters also need to select a category, either compliment, neutral, or complaint. With quantifiable social feedback, different preferences can be *identified* and measured by cluster analysis.

Once conflicting user preferences can be detected, the question remains how to *resolve* the conflicts. To set a starting point for future research, we investigated the effects of applying group recommendation techniques [147, 148, 193] to requirements negotiation in an empirical study [99]. Our results show that group recommendations and the visibility of preferences to other stakeholders have an impact on the perceived usability and quality of the provided decision support. Moreover, our results indicate that the *majority rule,* which takes decisions according to the majority of stakeholders' votes, is a simple but effective heuristic in requirements negotiation – a finding which is confirmed by related work [133].

#### 7.3.2.3 Social Configuration Management

Software systems need to evolve over time to remain usable and relevant to their users [113, 171]. Recent software development processes embrace this change. Configuration management [142] is concerned with managing and controlling change during software evolution [49]. To enable developers to deal with change, it introduces a formal process to capture, analyze, and work on *change requests*, formal reports created by users or developers and denote the request to modify a work product [49]. In particular, configuration management needs to deal with different, possibly coexisting, *versions* of a specific software. While some versions (called *variants*) are intended to coexist, others do for instance due to users' unwillingness or inability to update to newer baselines. Especially mobile devices' capabilities seemingly "deteriorate" quickly. New operating system versions often do not run on older devices, so that a software version relying on new API calls cannot be installed by users of these devices. In practice this leads to a mixed user audience owning different devices and, as a consequence, more and more different running software versions. A recent study [77] drastically demonstrates the whole extent of this issue. The author investigated the history of operating system updates on Android phones and revealed that 7 of 18 devices never ran a current version, while 12 only ran a current version for up to a couple of weeks.

Portneuf connects different users who have similar user experiences and by that means forms user communities. The framework's lowest layer consists of monitoring facilities, which collect events about the application and its context. Whenever a user wants to provide feedback, recommendations for existing feedback are calculated based on a comparison of this monitored usage data. The application version is part of the information collected by Portneuf and also gets reported to the development team. However, at this point ends Portneuf's support for managing software configuration.

We claim that an integration of configuration management knowledge and user involvement during software evolution is worth further investigation in both theoretical (e.g. integration of meta-models) and practical (e.g. improvement of configuration management tools) aspects. We see three possible extensions to the current state of the framework which should be investigated by future work.

**Integration of User Feedback** As our interviews with software professionals (cf. Chapter 3) show, developers need longitudinal tracking facilities for user feedback. User feedback should be seen as an important development artifact, which is often the source of design rationale, and needs to be visualized and traceable throughout the whole software lifecycle. Issue trackers which provide facilities to track problems, in particular error reports, already include fields to capture the version for which a report applies. In contrast, we argue that user feedback should be regarded as long-term artifacts which span multiple software versions, more similar to user requirements specifications. Future work should therefore investigate more thoroughly, which requirements developers have regarding the integration of user feedback into existing processes, tools, and specifications, and how configuration management can be supported.

**Cross-Version User Feedback Recommendation** Future studies should analyze the impact of comparing user experience from different versions of the same software. While Portneuf does not limit its recommendations to users of the same version, the impact of this approach is unknown. Further, configuration management knowledge bases could be used (a) to inform users about changes in other versions which may correspond to their feedback and (b) to inform developers about possibly re-occurring problems, similar to re-opening a ticket in an issue tracker.

**Social Beta** Current software projects typically follow one of three release management approaches. *First* and most commonly, new versions are defined based on release plans, which identify enhancements and improvements that will be included. Before a major release, developers typically create pre-release versions, which might also be distributed to dedicated users, so called *beta-testers*. Their feedback helps to perfect the upcoming release, but does not change the

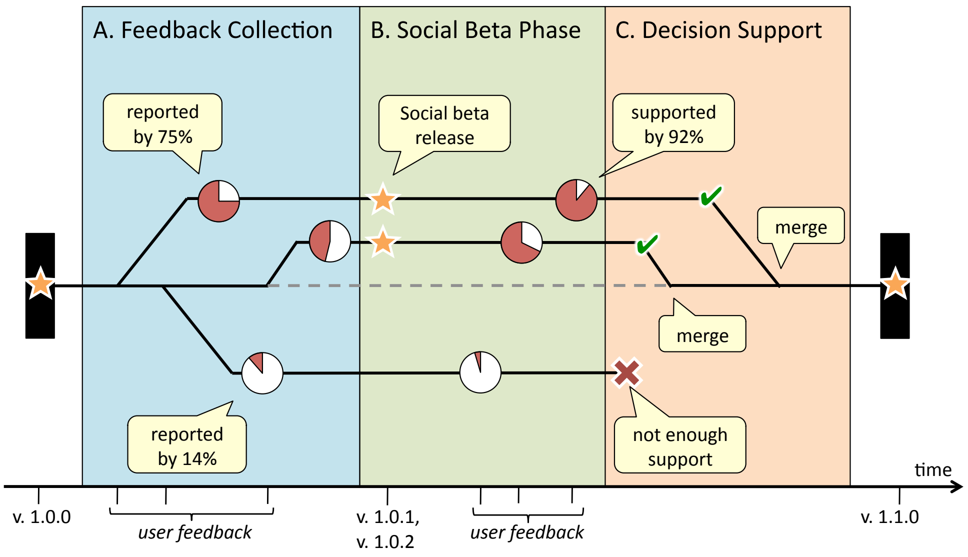


Figure 7.2: Sample social beta release tree.

release plan. *Second*, *software product lines* [233] are special software variants, which are created by configuring reusable software assets. To this end, software product lines engineering identifies *commonalities* and *variabilities* of a product within a specific application domain and captures them in product family specifications. Creating a new product line then involves systematically deriving a new configuration from the product family and combining the existing assets accordingly. The two main advantages of this approach are low product development costs and the possibility to serve multiple target groups. *Third*, in the *perpetual beta* paradigm [215] a product is incrementally improved and enhanced while already being rolled out. What was sarcastically called *bananaware* before the days of Web 2.0, has now become a mainstream approach to software development – especially for web applications.

We argue that these three approaches might be combined beneficially to a *"social beta"* approach when collective user feedback is available. In the social beta approach, clusters of user feedback determine small incremental improvements and enhancements. As shown in Figure 7.2, beta versions incorporating the corresponding changes might be released simultaneously, similar to small temporary product lines. Based on the impact of the test results, which again take the form of user feedback, the different branches can then be merged before the next major release.

In contrast to Ali et al. [7], who proposed *social software product lines* based on user-guided adaptation, we suggest to utilize multiple parallel beta versions to evaluate the impact of small improvements and enhancements. In particular in large software projects, there might be conflicting preferences regarding the system's functionality among all stakeholders [99, 217]. Moreover, research has shown that stakeholders' preferences do not remain stable [98]. The social beta approach allows developers to try different possibly contradictory preferences in parallel, to collect measures about the impact of each, and to decide for and release the most impactful package.

Consequently, future research should evaluate the social beta approach in real software projects. Specifically, we see three categories of research questions which are relevant in this context. *First*, in which situations is it reasonable to involve users in a social beta fashion? *Second*, which are the project management and infrastructure requirements to realize such an approach? *Third*, does the social beta approach add value from the software developers' and managers' point of view?

## 7.3.3 User Experience Applications

User experience is inherently a conceptualization of information about the interplay between users and applications in a specific context over a particular time period. While we have shown how to harness user experience as enabler for continuous user involvement in software engineering activities, this application is no end in itself. We summarize three additional applications of user experience in other domains, highlighting relevant research problems and suggesting directions for appropriate solutions.

### 7.3.3.1 Question and Answer Sites

On question and answer sites such as StackOverflow[2], developers can ask questions on almost any technical area, and typically receive answers within few minutes [189]. But StackOverflow has already left behind the state of being a simple forum, evolving into a community based encyclopedia of development knowledge: Research has shown that considerable parts (up to 87%) of public APIs are documented by the crowd [226, 227]. However, "it is difficult to answer, when one does not understand the question"[3] . Accordingly, it depends on the question if and how quickly developers find answers when searching Q&A sites. While it might be straightforward to look up answers for a specific exception, things get more complicated when verbalizing more complex user and application behavior. For instance, Chilana et al. [66] propose a new approach to help, which lets users choose a user interface element that they believe is relevant to their problem. Likewise, we think that in particular question and answer sites made for *users*, such as SuperUser[4] or DrupalAnswers[5] can benefit

[2]http://stackoverflow.com
[3]Quotation from ambassador *Sarek* in "Star Trek IV".
[4]http://superuser.com/
[5]http://drupal.stackexchange.com/

from a more efficient search mechanism and eventually from being embedded into applications.

We envision that questions are enriched by characteristic user experience snapshots, what provides two benefits. First, other users can look up relevant answers to problems they currently experience rather than searching for them. To this end, their user experience snapshot needs to be sent to the Q&A site. Then, the Portneuf recommender subsystem can select a list of relevant questions and answers. As a result, users might find answers without the need to formulate a question. Moreover, the recommended answers might be presented as contextual help, so that users do not even need to leave the running application. Second, moderators with experience in specific subjects can be directly pointed to open questions in their area of expertise. To this end, future research needs to explore ways to symbolize user experience, so that moderators might register for questions which include specific events.

#### 7.3.3.2 Instrumented Spaces and Embedded Systems

Instrumented spaces and environments are an infrastructure for ubiquitous computing [95]. Typical instrumented spaces are equipped with sensors and actuators, which allow for monitoring and acting on otherwise uninvolved physical systems. Application scenarios include specialized environments for people who need constant assistance such as elders [16, 280] and environments whose goal is to reduce wasting of energy [129]. The building sector has a particularly high potential to address climate change. For instance, more than 65% of all electricity produced in the U.S. is used for commercial and residential buildings [175]. Instrumented spaces such as intelligent workplaces [128, 270] aim at improving the energy footprint of commercial buildings. To this end, research explores the effects of automating building fixtures and utilizing intelligent control strategies such as occupancy sensing and daylight harvesting on overall energy consumption. Mobile and ubiquitous systems facilitate the complementary involvement of users by according them personal control over utility fixtures, which is important for their acceptance of intelligent workplaces and provides for sustained energy efficiency [231].

For instance, intelligent systems might suggest a user to adjust her computer's power settings after observing a particularly long idle period. Thus, the limits of individual control in intelligent workplaces are determined in particular by common energy saving rules or constraints. In other words, intelligent workplaces are complex ecosystems characterized by an interplay of hardware, software, and groups of human users with individual needs and common constraints. In order to fine-tune common constraints or at least inform their administrators, users are encouraged to provide feedback on the suggestions given by the intelligent workplace. The result of these interactions is a large amount of feedback on suggestions which were given to specific users in a particular context. But to react on this feedback, administrators of the intelligent workplace or developers of underlying software need to understand the conditions under which feedback was given, and have to be able to assess how many users are affected by specific change requests.

Portneuf utilizes monitoring data about the use of software in a specific context to consolidate user feedback and allow developers to assess its importance in the user community. For the purpose of this dissertation, we focused on typical interactive software and deliberately excluded hardware sensors to reduce complexity. However, our definition of context as well as its conceptual representation in the Portneuf model, explicitly includes the notion of *environment*. Moreover, while we concentrated on users in the form of actual persons, conceptually also *embedded systems* might become users of an application. We claim that future research on instrumented spaces should apply Portneuf to gather contextual information about the environment as part of user experience when collecting feedback on intelligent suggestions and actuators. We believe that context-sensitive user experience is helpful information for administrators and developers of intelligent workplaces and might serve as decision support when adjusting or learning more individual energy saving rules. Moreover, future work should investigate if and how embedded systems can take the role of users in the described ecosystems, and in which way their feedback can influence how participating components are controlled.

#### 7.3.3.3 Corrective Maintenance and User Comprehension

Error reports typically lack context information which allows developers to understand the conditions under which a specific error happened [298]. Especially the necessary steps to reproduce a reported error are often missing, incomplete, or ambiguous. To this end, software maintenance research investigates methods to automatically collect the relevant information. Existing approaches range from the automatic collection of stack traces to issue trackers guiding reporters while submitting errors [298].

However, stack traces alone do not really allow developers to understand what a specific user did and in which particular context, as they represent only a single snapshot of the application's reaction. Therefore, research aims at automatizing the reproduction of more complex and dynamic user behavior. For instance, we developed an error replayer which accompanies a video-like representation of users' interactions with relevant contextual information [219]. The system, which is part of the FastFix research project[6], allows for lightweight, deterministic fault replication of concurrent programs by using cooperative recording and partial log combination [185].

[6]http://www.fastfixproject.eu

To further improve *user comprehension*, we propose to visualize the user experience collected by PORTNEUF to support developers while reproducing errors. User experience snapshots contain context-sensitive information about user, application, and environment, which might be helpful to understand what users did to obtain a specific error, and why. The FastFix project mentioned above as well as tools like TimeLine [122], which aim at visualizing steps to reproduce might be a starting point for future research. In addition, we suggest to distinguish between *short-term* and *long-term user experience*. Similar to short-term and long-term interests [27], we claim that user experience will have *peaks* and *trends*. Peaks (short-term) might determine a specific scenario and help to understand a particular situation, while trends (long-term) give evidence on clusters of users or typical feature sets.

A particular challenge lies in feedback on mobile user experience: Given the small screen estate of mobile devices, feedback on the usability of a mobile application is different from classical desktop applications [140]. While the latter might be concerned with the arrangement of the user interface, mobile applications typically present only few user interface items but on *multiple screens*. As a consequence, feedback on the usability of mobile applications will rather concern *navigational paths* than single user interface elements. We think that future research needs to explore, as first step, how users can be supported when providing feedback on mobile applications. Which methods do they need to explain their pains, which techniques to record and present shortcomings in navigation paths?

Finally, future work should investigate methods how to detect usability issues in mobile applications by analyzing large amounts of user experience data. To provide a starting point for this work, we developed a heuristic method which detects so-called *low discoverability* usability errors in mobile applications [14]. To this end, we analyze monitored sequences of users' transitions between the views of a specific mobile application using sequential pattern mining. We found a specific navigational pattern, characterized by *low retention* and *navigational loops*, that is an indicator for low discoverability errors. We performed a preliminary evaluation study which shows that our method has the potential to be applied to mobile applications in general.

# Appendix A

# Interview Questions

The following sections include our interview questions for analyzing user involvement in software evolution practice (cf. Chapter 3). We conducted semi-structured, open-ended interviews, which allow for improvisation and thus facilitate an exploration of the studied cases. To this end, we provided each subject with the question catalogue one week before the interview. We explicitly called their attention to the semi-structured nature of the interview, and underlined that the questions should be regarded as support. If a subject was working in multiple projects, we asked her to select one project which was developed for a large number of end users.

## A.1 Project Information

1. Of which **type** is the **project**?
    - ( ) closed source,
    - ( ) open source,
    - ( ) other:
2. Which **kind of software** is the project? For instance,
    - [ ] mobile application,
    - [ ] desktop application,
    - [ ] off the shelf-component,
    - [ ] customized software,
    - [ ] other:
3. Which **user audience** do you target?
    - ( ) any user,
    - ( ) special group of users:
    - ( ) other:

4. **How many active users** does your software have?
    - ( ) less than 100: ___
    - ( ) between 100 and 500
    - ( ) between 500 and 1.000
    - ( ) between 1.000 and 5.000
    - ( ) between 5.000 and 10.000
    - ( ) more than 10.000: ____
    - ( ) no answer
5. How frequently do you **release** a new external version?
    - ( ) more often than every 2 weeks,
    - ( ) every 2 to 4 weeks,
    - ( ) every 4 to 8 weeks,
    - ( ) every 2 to 6 months,
    - ( ) less often than every 6 months

## A.2 User Feedback - Current Landscape

1. How can your users **contribute** to improve your software?
    - [ ] by reporting errors
    - [ ] by requesting new features
    - [ ] by providing feedback on existing features
    - [ ] by requesting improvements or enhancements
    - [ ] by rating the product
    - [ ] other:
2. How often can your users **contribute** to improve your software?
    - [ ] at the beginning of a project (e.g. by focus groups)
    - [ ] regularly (e.g. before you plan a new major release)
    - [ ] continuously (i.e. at any time during the software lifecycle)
    - [ ] other:
3. How do your users **report errors**? (**o**ften, **s**ometimes, **r**arely, **n**ever)
    - [ ] by using a public issue tracker,

- [ ] by sending email,
- [ ] by calling on the phone,
- [ ] by writing blog posts,
- [ ] by writing to mailing-lists,
- [ ] by publishing their opinion on other publicly available media,
- [ ] by using a reporting mechanism which is integrated in the software,
- [ ] by using a specialized service (e.g. user-voice, get-satisfaction),
- [ ] other:

4. How do your users **request new features**? (**o**ften, **s**ometimes, **r**arely, **n**ever)

   - [ ] we actively carry out studies with a representative group,
   - [ ] by using a public issue tracker,
   - [ ] by sending email,
   - [ ] by calling on the phone,
   - [ ] by writing blog posts,
   - [ ] by writing to mailing-lists,
   - [ ] by publishing their opinion on other publicly available media,
   - [ ] by using a feedback mechanism which is integrated in the software,
   - [ ] by using a specialized service (e.g. user-voice, get-satisfaction),
   - [ ] other:

5. How do your users provide **feedback on existing features**? (**o**ften, **s**ometimes, **r**arely, **n**ever)

   - [ ] we actively carry out studies with a representative group,
   - [ ] by using a public issue tracker,
   - [ ] by sending email,
   - [ ] by calling on the phone,
   - [ ] by writing blog posts,
   - [ ] by writing to mailing-lists,
   - [ ] by publishing their opinion on other publicly available media,
   - [ ] by using a feedback mechanism which is integrated in the software,
   - [ ] by using a specialized service (e.g. user-voice, get-satisfaction),
   - [ ] other:

6. How do your users request **enhancements and improvements** to your software? (**o**ften, **s**ometimes, **r**arely, **n**ever)
    - [ ] we actively carry out studies with a representative group,
    - [ ] by using a public issue tracker,
    - [ ] by sending email,
    - [ ] by calling on the phone,
    - [ ] by writing blog posts,
    - [ ] by writing to mailing-lists,
    - [ ] by publishing their opinion on other publicly available media,
    - [ ] by using a feedback mechanism which is integrated in the software,
    - [ ] by using a specialized service (e.g. user-voice, get-satisfaction),
    - [ ] other:

## A.3 User Feedback - Current Workflow and Problems

1. Why is user feedback **interesting** for you?
    - [ ] it helps to improve the software quality (e.g. to fix bugs, to improve features)
    - [ ] it helps to find missing features
    - [ ] it helps to understand what users want and need
    - [ ] it helps to advertise and market the application
    - [ ] it helps to understand, if and how the product is accepted (e.g. by ratings)
    - [ ] other:
2. How do you **process** user feedback, why, and which problems do you encounter?
3. Which activities **take most time** while processing user feedback?
4. Which activities **are the most difficult** while processing user feedback?
5. Do you **relate single reports** to each other (e.g. to identify duplicates or antipodes)?
    - ( ) yes, these get mapped automatically by a tool

- ( ) yes, our users correlate their feedback themselves
- ( ) yes, we do this manually before starting the analysis
- ( ) yes, we do this successively while analyzing the single reports
- ( ) no, we don't. Reason:

6. **How easy** or intuitive is it for you to relate different feedback to each other?
   - ( ) very easy
   - ( ) somewhat easy
   - ( ) undecided
   - ( ) somewhat difficult
   - ( ) very difficult
7. **According to what** do you relate single reports?
   - [ ] if both users had the same experience with the software
   - [ ] if the reports are duplicates
   - [ ] if the reports are antipodes
   - [ ] if one feedback details another
   - [ ] if the reports concern the same feature
   - [ ] if the type of the reports is the same (e.g. both are feature requests)
   - [ ] other:
8. Overall, **how satisfied** are you with your current practice of processing user feedback?
   - ( ) very satisfied
   - ( ) somewhat satisfied
   - ( ) undecided
   - ( ) somewhat unsatisfied
   - ( ) very unsatisfied

## A.4 User Feedback - Potentials and Challenges

1. Would you embrace **tool support** to **relate** user feedback to each other and why?
2. **When** should user feedback be related by a tool

- [ ] the tool should show the user relevant existing feedbacks to allow her to comment on, rate, or vote for them (proactive)
- [ ] the tool should simply collect user feedback. The gathered reports should be analyzed and related later (reactive)
- [ ] other:

3. **According to what** should user feedback be related by a tool? (open question)
   - [ ] if the users had the same experience with the software
   - [ ] if the reports are duplicates
   - [ ] if the reports are antipodes
   - [ ] if one feedback details another
   - [ ] if the reports concern the same feature
   - [ ] if the type of the reports is the same (e.g. both are feature requests)
   - [ ] other:
4. Is it important for you to assess the **potential** of a specific user feedback **to improve your software** and why?
   a) How would you **assess** the **importance** of a specific user feedback for **your project**?
      i. [ ] by the quantity of its occurrence,
      ii. [ ] by assessing the individual user who contributed,
      iii. [ ] only after working on it (retrospective),
      iv. [ ] not at all. Reason:
      v. [ ] other:

   b) Would you embrace **tool support** for this assessment and why?
5. Is it important for you to assess the **importance** of a specific user feedback **to the user community** and why?
   a) How would you **assess** the **importance** of a specific user feedback to the **user community**? (multiple answers allowed)
      i. [ ] by the quantity of its occurrence,
      ii. [ ] by assessing the individual user who contributed,
      iii. [ ] only after working on it (retrospective),
      iv. [ ] not at all. Reason:

        v. [ ] other:

   b) Would you embrace **tool support** for this assessment and why?

6. Can you tell if there are **specific individual users** that provide particularly important or frequent information, and how?
   a) Do you or would you **treat their feedback differently**
        i. [ ] yes, we read it first
        ii. [ ] yes, if we have a lot of similar feedback, we only / first read the feedback of this user
        iii. [ ] no, we don't. Reason:
        iv. [ ] other:
   b) Do you or would you let the users or the **community know that they are important** and how?
        i. [ ] yes, we let important users know that their feedback is important to us
        ii. [ ] yes, we let a specific user know if a specific feedback was important to us
        iii. [ ] yes, we let the community know which users are important to us
        iv. [ ] yes, we let the community know which feedback is important to us
        v. [ ] no, we don't. Reason:

## A.5 Personal Information

1. How would you describe **your role** in the project?
   - [ ] requirements engineer
   - [ ] developer
   - [ ] tester
   - [ ] architect
   - [ ] project manager
   - [ ] other:
2. How many years of **experience** do you have in your main role?
   - ( ) less than 1 year,

- ( ) between 1 and 2 years,
- ( ) between 3 and 5 years,
- ( ) between 6 and 10 years,
- ( ) more than 10 years

# Appendix B
# List of Portneuf Sensors

Table B.1 lists the sensors we implemented for our summative evaluation with BetterTouchTool. Sensors 1 through 8 are generic, application independent sensors for Mac OS X and iOS, while sensors 9 and 10 are application-specific.

Table B.1: Generic and application-specific PORTNEUF sensors.

| # | monitored event type | sensor type | added in version |
|---|---|---|---|
| 1 | exceptions | generic | 0.799 |
| 2 | signals | generic | 0.799 |
| 3 | console logs | generic | 0.799 |
| 4 | user interactions | generic | 0.799 |
| 5 | action messages | generic | 0.799 |
| 6 | application switches | generic | 0.799 |
| 7 | wake, sleep, and quit events | generic | 0.91 |
| 8 | high memory usage | generic | 0.91 |
| 9 | gestures and shortcuts | application-specific | 0.799 |
| 10 | window snapping actions | application-specific | 0.799 |

# List of Figures

# List of Tables

# Bibliography

[1] Stack Overflow. Last accessed: 01.08.2012. Available from: `http://www.stackoverflow.com`.

[2] A. Abdul-Rahman and S. Hailes. Supporting Trust in Virtual Communities. In R. H. Sprague, editor, *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, pages 1–9, Hawaii, HI, USA, 2000. Published by the IEEE Computer Society, IEEE.

[3] ACM. ACM Digital Portal. Last accessed: 28.08.2012. Available from: `http://portal.acm.org/portal.cfm`.

[4] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *SIGMOD Conference on Management of Data*, pages 207–216, Washington, DC, USA, 1993. ACM.

[5] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering*, pages 3–14, Taipei, Taiwan, 1995. IEEE Comput. Soc. Press.

[6] L. Alben. Quality of Experience: Defining the Criteria for Effective Interaction Design. *Interactions*, 3(3):11–15, 1996.

[7] R. Ali, C. Solis, F. Dalpiaz, W. Maalej, P. Giorgini, and B. Nuseibeh. Social Software Product Lines. In *2011 First International Workshop on Requirements Engineering for Social Computing*, pages 14–17, Trento, Italy, Aug. 2011. IEEE.

[8] R. Ali, C. Solis, I. Omoronyia, M. Salehie, and B. Nuseibeh. Social Adaptation - When Software Gives Users a Voice. In *7th International Evaluation of Novel Approaches to Software Engineering (ENASE'12)*, pages 75–84, Wroclaw, Poland, 2012.

[9] S. Anand and B. Mobasher. Contextual Recommendation. *Discovering and Deploying User and Content Profiles*, pages 142–160, 2007.

[10] W. L. Anderson and W. T. Crocca. Engineering practice and codevelopment of product prototypes. *Communications of the ACM*, 36(6):49–56, June 1993.

[11] S. J. Andriole. *Storyboard prototyping: a new approach to user requirements analysis.* QED Information Sciences, Inc., Wellesley, MA, USA, Jan. 1989.

[12] J. Anvik, L. Hiew, and G. C. Murphy. Coping with an open bug repository. In *eclipse '05: Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange*, pages 35–39, San Diego, California, USA, 2005.

[13] Apple Inc. Apple App Store. Last accessed: 29.08.2012. Available from: `http://itunes.apple.com/de/genre/ios/id36?mt=8`.

[14] D. Bader and D. Pagano. Towards Automated Detection of Mobile Usability Issues. In *Proceedings of the First European Workshop on Mobile Engineering*, Aachen, Germany, 2013.

[15] D. Bajic and K. Lyons. Leveraging social media to gather user feedback for software development. In *Proceeding of the 2nd international workshop on Web 2.0 for software engineering*, pages 1–6, Honolulu, HI, USA, 2011. ACM.

[16] A. Bamis, D. Lymberopoulos, T. Teixeira, and A. Savvides. Towards precision monitoring of elders for providing assistive services. In *Proceedings of the 1st ACM international conference on PErvasive Technologies Related to Assistive Environments - PETRA '08*, pages 1–8, Athens, Greece, 2008. ACM.

[17] H. Barki and J. Hartwick. Rethinking the concept of user involvement. *MIS quarterly*, 13(1):53–64, 1989.

[18] H. Barki and J. Hartwick. User participation and user involvement in information system development. In *Proceedings of the Twenty-Fourth Annual Hawaii International Conference on System Sciences*, pages 487–492, Hawaii, USA, 1991.

[19] J. J. Baroudi, M. H. Olson, and B. Ives. An Empirical Study of the Impact of User Involvement on System Usage and Information Satisfaction. *Communications of the ACM*, 29(3):232–238, 1986.

[20] K. Battarbee. Defining co-experience. In *Proceedings of the 2003 international conference on Designing pleasurable products and interfaces - DPPI '03*, page 109, Pittsburgh, PA, USA, 2003. ACM.

[21] K. Beck and C. Andres. *Extreme Programming Explained: Embrace Change.* Addison-Wesley Reading, 2nd edition, 2005.

[22] G. Beenen, K. Ling, X. Wang, K. Chang, D. Frankowski, P. Resnick, R. E. Kraut, and A. Arbor. Using Social Psychology to Motivate Contributions to Online Communities. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 212–221, Chicago, Illinois, USA, 2004. ACM.

[23] A. Begel, R. DeLine, and T. Zimmermann. Social media for software engineering. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 33–38, Santa Fe, NM, USA, 2010. ACM.

[24] M. Bekker and J. Long. User Involvement in the Design of Human-Computer Interactions: Some Similarities and Differences between Design Approaches. In S. McDonald, Y. Waern, and G. Cockton, editors, *People and Computers XIV - Usability or Else!*, pages 135–147. Springer, 2000.

[25] B. Bergvall-Kåreborn and A. Ståhlbröst. Participatory Design - One Step Back or Two Steps Forward? In *Proceedings of the Tenth Anniversary Conference on Participatory Design 2008*, pages 102–111, Bloomington, IN, USA, 2008. ACM.

[26] R. G. Bias. The pluralistic usability walkthrough: coordinated empathies. In J. Nielsen and R. L. Mack, editors, *Usability inspection methods*, pages 63–76. John Wiley & Sons, Inc. New York, NY, USA, June 1994.

[27] D. Billsus and M. Pazzani. User Modeling for Adaptive News Access. *User Modeling and User-Adapted Interaction*, 10(2-3):147–180, 2000.

[28] W. Biscardi. FCPX: What Pros Find Missing in Final Cut Pro X. Last accessed: 08.12.2012, 2011. Available from: `http://magazine.creativecow.net/article/final-cut-pro-x-whats-missing-for-some-pros`.

[29] Bit Stadium GmbH. HockeyApp. Last accessed: 09.11.2012, 2012. Available from: `http://hockeyapp.net`.

[30] G. Bjerknes and T. Bratteteig. Florence in wonderland: System development with nurses. In G. Bjerknes, P. Ehn, and M. Kyng, editors, *Computers and democracy: A Scandinavian challenge*, pages 279–295. Gower, Brookfield, VT, 1987.

[31] G. Bjerknes and T. Bratteteig. User Participation and Democracy : A Discussion of Scandinavian Research on System Development. *Scandinavian Journal of Information Systems*, 7(1):73–98, 1995.

[32] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3(4-5):993–1022, May 2003.

[33] J. Blomberg, J. Giacomi, A. Mosher, and P. Swenton-Wall. Ethnographic field methods and their relation to design. In D. Schuler and A. Namioka, editors, *Participatory design: Principles and practices.* Erlbaum, Hillsdale, NJ, USA, 1993.

[34] J. Blomberg, L. Suchman, and R. Trigg. Back to work: renewing old agendas for cooperative design. In M. Kyng and L. Mathiassen, editors, *Computers and Design in Context*, chapter 10, pages 267–287. MIT Press, Cambridge, MA, USA, Nov. 1997.

[35] S. Bloomer. Real Projects Don't Need User Interface Designers: Overcoming the Barriers to HCI in the Real World. In *Proceedings of OZCHI93, the CHISIG Annual Conference on Human-Computer Interaction*, pages 94–108, Canberra, Australia, 1993. ACM.

[36] S. Bloomer, R. Croft, and L. Wright. Collaborative design workshops: a case study. *interactions*, 4(1):31–39, Jan. 1997.

[37] S. Bødker, E. Christiansen, and M. Thüring. A conceptual toolbox for designing CSCW applications. In *Proceedings of the First International Workshop on the Design of Cooperative Systems*, Antibes-Juan-Les-Pins, France, 1995. INRIA Press, Rocquencourt.

[38] S. Bødker, P. Ehn, M. Kyng, J. Kammersgård, and Y. Sundblad. A UTOPIAN experience: On design of powerful computer-based tools for skilled graphic workers. In G. Bjerknes, P. Ehn, and M. Kyng, editors, *Computers and democracy: A Scandinavian challenge*, pages 251–278. Gower, Brookfield, VT, USA, 1987.

[39] B. Boehm. Requirements that Handle IKIWISI, COTS, and Rapid Change. *IEEE Computer*, 33(7):99–102, 2000.

[40] B. Boehm and R. Ross. Theory W Software Project Management: Principles and Examples. *IEEE Transactions on Software Engineering*, 15(7):902–916, 1989.

[41] B. W. Boehm. A Spiral Model of Software Development and Enhancement. *IEEE Computer*, (May):61–72, 1988.

[42] R. Bonacin and M. C. C. Baranauskas. Usability in the Organisational Context: a Semiotic-Participatory Approach. In *The 7th International Workshop on Organisational Semiotics*, Setúbal, Portugal, 2004.

[43] G. A. Boy. The group elicitation method for participatory design and usability testing. *interactions*, 4(2):27–33, Mar. 1997.

[44] K. Braa. Influencing System Quality by Using Decision Diaries in Prototyping Projects. In *PDC '92: Proceedings of the Participatory Design Conference*, pages 163–170, Cambridge, MA, USA, 1992.

[45] K. Braa. Priority workshops: springboard for user participation in redesign activities. In *Proceedings of conference on Organizational computing systems - COCS '95*, pages 258–267, Milpitas, CA, USA, Aug. 1995. ACM Press.

[46] K. Braa, T. Bratteteig, J. Kaasboll, and L. Ogrim. ENtry to the FIRE Project. Technical report, University of Oslo, Department of Informatics, Oslo, 1992.

[47] P. J. Brown, J. D. Bovey, and C. Xian. Context-aware applications: from the laboratory to the marketplace. *IEEE Personal Communications*, 4(5):58–64, 1997.

[48] B. Bruegge. *Adaptability and Portability of Symbolic Debuggers.* Dissertation, Carnegie-Mellon University, 1985. Available from: `http://reports-archive.adm.cs.cmu.edu/anon/scan/CMU-CS-85-174.pdf`.

[49] B. Bruegge and A. H. Dutoit. *Object-Oriented Software Engineering Using UML, Patterns and Java.* Prentice Hall, 3rd edition, 2010.

[50] B. Bruegge, D. Harhoff, A. Picot, O. Creighton, M. Fiedler, and J. Henkel. *Open-Source-Software: Eine ökonomische und technische Analyse.* Springer Berlin Heidelberg, 2004.

[51] R. Budde, K. Kuhlenkamp, L. Mathiassen, and L. Zullighoven, editors. *Approaches to Prototyping.* Springer Verlag, Berlin, Germany, Jan. 1984.

[52] R. Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002.

[53] D. T. Campbell and J. Stanley. *Experimental and Quasi-Experimental Designs for Research.* Wadsworth Publishing, 1963.

[54] M. Carbone, M. Nielsen, and V. Sassone. A Formal Model for Trust in Dynamic Networks. In *Proceedings of the First International Conference on Software Engineering and Formal Methods*, pages 54–61, Brisbane, Australia, 2003. IEEE, IEEE.

[55] S. K. Card, A. Newell, and T. P. Moran. *The Psychology of Human-Computer Interaction.* L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1983.

[56] J. M. Carroll, editor. *Scenario-based design: envisioning work and technology in system development.* John Wiley & Sons, Inc. New York, NY, USA, Sept. 1995.

[57] CASA Research Center - University of Massachusetts Amherst. End User Integration. Last accessed: 29.08.2012. Available from: `http://www.casa.umass.edu/main/research/end_user_integration/`.

[58] M. Castro, M. Costa, and J.-P. Martin. Better bug reporting with better privacy. In *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, volume 43, pages 319–328, Seattle, WA, USA, Mar. 2008. ACM.

[59] D. Centola. The spread of behavior in an online social network experiment. *Science*, 329(5996):1194–1197, Sept. 2010.

[60] L. Cerejo. The Elements Of The Mobile User Experience. In *Smashing Magazine*. Last accessed: 21.07.2012, 2012. Available from: `http://mobile.smashingmagazine.com/2012/07/12/elements-mobile-user-experience/`.

[61] R. Chandy and H. Gu. Identifying spam in the iOS app store. In *Proceedings of the 2nd Joint WICOW/AIRWeb Workshop on Web Quality - WebQuality '12*, pages 56–59, Lyon, France, 2012. ACM.

[62] N. Chapin, J. Hale, K. Khan, J. Ramil, and W. Tan. Types of software evolution and software maintenance. *Journal of software maintenance and evolution: Research and Practice*, 13(1):3–30, 2001.

[63] P. Checkland. *Systems thinking, systems practice.* Wiley Press, New York, USA, 1981.

[64] M. Chen and X. Liu. Predicting Popularity of Online Distributed Applications: iTunes App Store Case Analysis. In *Proceedings of the 2011 iConference*, pages 661–663, Seattle, WA, USA, 2011. ACM.

[65] H. W. Chesbrough. *Open Innovation: The New Imperative for Creating and Profiting from Technology*, volume 20 of *Praeger special studies in U.S. economic, social, and political issues.* Harvard Business School Press, 2003.

[66] P. K. Chilana, A. J. Ko, and J. O. Wobbrock. LemonAid : Selection-Based Crowdsourced Contextual Help for Web Applications. In *CHI'12*, pages 1549–1558, Austin, Texas, USA, 2012. ACM.

[67] J. Clause and A. Orso. Camouflage: Automated Sanitization of Field Data. Technical report, Georgia Institute of Technology, 2009.

[68] J. Cohen. *Statistical Power Analysis for the Behavioral Science.* Routledge, 2nd edition, 1988.

[69] A. Cooper. *The Inmates Are Running the Asylum.* Sams Publishing, Mar. 1999.

[70] J. Corbin and A. Strauss. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative sociology*, 13(1), 1990.

[71] D. Crane. Graphic recording in systems design. In *PDC '90: Conference on Participatory Design*, Seattle, WA, USA, 1990.

[72] J. W. Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches.* Sage Publications, Inc, 3rd edition, 2008.

[73] B. Dagenais and M. Robillard. Creating and evolving developer documentation: understanding the decisions of open source contributors. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pages 127–136, Santa Fe, NM, USA, 2010. ACM.

[74] L. Damodaran. User involvement in the systems design process - a practical guide for users. *Behaviour & Information Technology*, 15(6):363–377, 1996.

[75] C. R. B. de Souza and D. F. Redmiles. An empirical study of software developers' management of dependencies and changes. In *Proceedings of the 13th international conference on Software engineering - ICSE '08*, pages 241–250, Leipzig, Germany, 2008. ACM Press.

[76] G.-J. De Vreede and H. G. Sol. Combating organized crime with groupware: Facilitating user involvement in information system development. In D. Coleman, editor, *Proceedings of GroupWare '94 Europe*, San Mateo, CA, USA, 1994. Morgan Kauffman.

[77] M. Degusta. Android Orphans: Visualizing a Sad History of Support. Last accessed: 18.09.2012, 2011. Available from: `http://theunderstatement.com/post/11982112928/android-orphans-visualizing-a-sad-history-of-support`.

[78] M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems*, 22(1):143–177, Jan. 2004.

[79] S. Deterding, D. Dixon, R. Khaled, and L. Nacke. From Game Design Elements to Gamefulness: Defining "Gamification". In *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, pages 9–15, Tampere, Finland, 2011. ACM.

[80] J. Devietti, B. Lucia, L. Ceze, and M. Oskin. DMP: deterministic shared memory multiprocessing. In *ASPLOS '09*, pages 85–96, Washington, DC, USA, 2009. ACM.

[81] A. K. Dey. Context-aware computing: The CyberDesk project. In *AAAI 1998 Spring Symposium on Intelligent Environments*, pages 51–54, Palo Alto, CA, USA, 1998. AAAI Press.

[82] A. K. Dey and G. D. Abowd. Towards a Better Understanding of Context and Context-Awareness. In *Proceedings of the CHI 2000 Workshop on The What, Who, Where, When, and How of Context-Awareness*, pages 1–12, The Hague, Netherlands, 2000.

[83] J. B. Disbrow and B. Igou. Application Software: Maintenance and Support Guidelines. Technical report, Gartner Research, 2008. Available from: `http://www.gartner.com/id=844119`.

[84] S. M. Dray. Understanding and supporting successful group work in software design: Lessons from IDS. In *Proceedings of the Computer Supported Cooperative Work (CSCW '92) Conference*, Toronto, Canda, 1992. ACM.

[85] E. A. Dykstra and R. P. Carasik. Structure and support in cooperative environments: the Amsterdam Conversation Environment. *International Journal of Man-Machine Studies*, 34(3):419–434, Mar. 1991.

[86] K. D. Eason. User-centred design: for users or by users? *Ergonomics*, 38(8):1667–1673, Aug. 1995.

[87] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian. Selecting Empirical Methods for Software Engineering Research. In F. Shull, J. Singer, and D. I. K. Sjøberg, editors, *Guide to Advanced Empirical Software Engineering*, pages 285–311. Springer-Verlag London, 2008.

[88] A. Eastwood. Firm fires shots at legacy systems. *Computing Canada*, 19(2):17, 1993.

[89] eBay Inc. eBay. Last accessed: 01.08.2012. Available from: `http://www.ebay.com`.

[90] Edgewall Software. Ticket Types. Last accessed: 13.09.2012, 2012. Available from: `http://trac.edgewall.org/wiki/TicketTypes`.

[91] P. Ehn and M. Kyng. Cardboard computers: mocking-it-up or hands-on the future. In *Design at work: Cooperative design of computer systems*, pages 169–196. Erlbaum, Hillsdale, NJ, USA, Jan. 1992.

[92] P. Ehn and D. Sjögren. From system descriptions to scripts for action. In *Design at work: Cooperative design of computer systems*, pages 241–268. Erlbaum, Hillsdale, NJ, USA, Jan. 1992.

[93] P. Elmer-DeWitt. 600 filmmakers sign complaint about Final Cut Pro X. Last accessed: 08.12.2012, 2011. Available from: `http://tech.fortune.cnn.com/2011/06/27/600-filmmakers-sign-complaint-about-final-cut-pro-x/`.

[94] Elsevier. ScienceDirect. Last accessed: 28.08.2012, 2012. Available from: `http://www.sciencedirect.com`.

[95] C. Endres, A. Butz, and A. MacWilliams. A Survey of Software Infrastructures and Frameworks for Ubiquitous Computing. *Mobile Information Systems*, 1(1):41–80, 2005.

[96] E. Enkel, J. Perez-Freije, and O. Gassmann. Minimizing Market Risks Through Customer Integration in New Product Development: Learning from Bad Practice. *Creativity and Innovation Management*, 14(4):425–437, Dec. 2005.

[97] A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker. An Environment for the Development of Knowledge-based Recommender Applications. *International Journal of Electronic Commerce (IJEC)*, 11(2):11–34, 2006.

[98] A. Felfernig, M. Schubert, M. Mandl, F. Ricci, and W. Maalej. Recommendation and decision technologies for requirements engineering. In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering - RSSE '10*, pages 11–15, Cape Town, South Africa, 2010. ACM.

[99] A. Felfernig, C. Zehentner, G. Ninaus, H. Grabner, W. Maalej, D. Pagano, L. Weninger, and F. Reinfrank. Group Decision Support for Requirements Negotiation. In L. Ardissono and T. Kuflik, editors, *Advances in User Modeling - Lecture Notes in Computer Science Volume 7138*, volume 7138 of *Lecture Notes in Computer Science*, pages 105–116. Springer Berlin Heidelberg, 2012.

[100] R. T. Fielding and R. N. Taylor. Principled design of the modern Web architecture. *ACM Transactions on Internet Technology*, 2(2):115–150, May 2002.

[101] C. Floyd. STEPS - a methodical approach to PD. *Communications of the ACM*, 36(6):83, June 1993.

[102] B. Flyvbjerg. Five Misunderstandings About Case-Study Research. *Qualitative Inquiry*, 12(2):219–245, Apr. 2006.

[103] J. Forlizzi and K. Battarbee. Understanding experience in interactive systems. In *Proceedings of the 2004 conference on Designing interactive systems processes, practices, methods, and techniques - DIS '04*, pages 261–268, Cambridge, MA, USA, 2004. ACM.

[104] J. Forlizzi and S. Ford. The Building Blocks of Experience: An Early Framework for Interaction Designers. In *Proceedings of the 3rd conference on Designing Interactive Systems: processes, practices, methods, and techniques (DIS '00)*, pages 419–423, Brooklyn, NY, USA, 2000. ACM.

[105] S. T. J. Foster and C. R. Franz. User involvement during information systems development: a comparison of analyst and user perceptions of system acceptance. *Journal of Engineering and Technology Managament*, 16(1999):329–348, 1999.

[106] Foursquare Labs Inc. Foursquare. Last accessed: 01.08.2012.

[107] J. Froehlich, M. Y. Chen, S. Consolvo, B. Harrison, J. A. Landay, S. B. Street, and S. Mateo. MyExperience: A System for In situ Tracing and Capturing of User Feedback on Mobile Phones. In *Proceedings of the 5th international conference on Mobile systems, applications and services (MobiSys '07)*, pages 57–70, San Juan, Puerto Rico, 2007. ACM.

[108] J. J. Garrett. *The Elements of User Experience: User-Centered Design for the Web.* New Riders, 2003.

[109] S. Gärtner and K. Schneider. A Method for Prioritizing End-User Feedback for Requirements Engineering. In *5th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 47–49, Zurich, Switzerland, 2012. IEEE.

[110] Get Satisfaction Inc. Online Community Software. Last accessed: 8.4.2012. Available from: `http://getsatisfaction.com`.

[111] B. G. Glaser and A. L. Strauss. *The Discovery of Grounded Theory: Strategies for Qualitative Research.* Aldine Transaction, 1967.

[112] J. Godfrey, M. W. Reed, and E. W. Herndon. Apps Across America - The Economics and Ecosystem of the Mobile App Market. Technical report, Association for Competitive Technology, Washington, DC, USA, 2012. Available from: `http://actonline.org/files/Apps-Across-America.pdf`.

[113] M. W. Godfrey and D. M. German. The past, present, and future of software evolution. In *Frontiers of Software Maintenance*, pages 129–138, Beijing, China, Sept. 2008. IEEE.

[114] I. Goldberg, D. Wagner, R. Thomas, and E. Brewer. A Secure Environment for Untrusted Helper Applications (Confining the Wily Hacker). In *Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography*, pages 1–13, San Jose, CA, USA, 1996. ACM.

[115] J. D. Gould and C. Lewis. Designing for Usability - Key Principles and What Designers Think. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 50–53, Boston, MA, USA, 1983. ACM.

[116] J. D. Gould and C. Lewis. Designing for Usability: Key Principles and What Designers Think. *Communications of the ACM*, 28(3):300–311, 1985.

[117] J. Greenbaum and K. H. Madsen. Small changes: Starting a participatory design process by giving participants a voice. In D. Schuler and A. Namioka, editors, *Participatory design: Principles and practices*. Erlbaum, Hillsdale, NJ, USA, 1993.

[118] K. Grønbæk and P. Mogensen. Specific Cooperative Analysis and Design in General Hypermedia Development. In *PDC '94: Proceedings of the Participatory Design Conference*, pages 159–171, Chapel Hill, North Carolina, USA, 1994.

[119] G. Grote. A participatory approach to the complementary design of highly automated work systems. In G. Bradley and H. W. Hendrik, editors, *Human factors in organizational design and management–IV*. Elsevier, Amsterdam, 1994.

[120] J. Grudin. Interactive systems: bridging the gaps between developers and users. *IEEE Computer*, 24(4):59–69, 1991.

[121] J. Grudin. Systematic Sources of Suboptimal Interface Design in Large Product Development Organizations. *Human-Computer Interaction*, 6(2):147–196, June 1991.

[122] N. Gurbanova. *Presenting User and Context Information to Developers during Bug Fixing*. Master's thesis, Technische Universität München, 2012.

[123] A. Guzzi, M. Pinzger, and A. van Deursen. Combining micro-blogging and IDE interactions to support developers in their quests. In *IEEE*

*International Conference on Software Maintenance (ICSM)*, pages 1–5, Timisoara, Romania, 2010. IEEE.

[124] E. V. Halpern. User Involvement in the Systems Analysis Functions. *ACM SIGCPR Computer Personnel*, 6(1-2), 1977.

[125] S. Harker. Using case studies in the iterative development of a methodology to support user-designer collaboration. In *INTERACT '93 and CHI '93 conference companion on Human factors in computing systems - CHI '93*, pages 57–58, Amsterdam, The Netherlands, Apr. 1993. ACM.

[126] M. Harman, Y. Jia, and Y. Zhang. App store mining and analysis: MSR for app stores. In *9th IEEE Working Conference on Mining Software Repositories (MSR)*, pages 108–111, Zurich, Switzerland, June 2012. IEEE.

[127] P. M. Hartigan and J. A. Hartigan. The dip test of unimodality. *Annals of Statistics*, 13(1):70–84, 1985.

[128] V. Hartkopf, V. Loftness, A. Mahdavi, S. Lee, and J. Shankavaram. An integrated approach to design and engineering of intelligent buildings-The Intelligent Workplace at Carnegie Mellon University. *Automation in Construction*, 6(5-6):401–415, Sept. 1997.

[129] V. Hartkopf, V. Loftness, P. Mill, and M. Siegel. Architecture and Software for Interactive Learning about Total Building Performance: Experience Based or Occupancy Expert Systems. In Y. Kalay, editor, *Evaluating and Predicting Design Performance*, chapter 11. John Wiley & Sons, Inc. New York, NY, USA, 1992.

[130] M. Hassenzahl. The Thing and I: Understanding the Relationship Between User and Product. In M. A. Blythe, K. Overbeeke, A. F. Monk, and P. C. Wright, editors, *Funology. From Usability to Enjoyment*, chapter 3, pages 31–42. Kluwer Academic Publishers, Dordrecht, Netherlands, 2003.

[131] M. Hassenzahl, A. Beu, and M. Burmester. Engineering Joy. *IEEE Software*, 18(1):70–76, 2001.

[132] M. Hassenzahl and N. Tractinsky. User experience - a research agenda. *Behaviour & Information Technology*, 25(2):91–97, Mar. 2006.

[133] R. Hastie and R. Kameda. The Robust Beauty of Majority Rules in Group Decisions. *Psychological Review*, 112(2):80–86, 2005.

[134] B. Hedberg. Computer systems to support industrial democracy. In E. Mumford and H. Sackman, editors, *Human Choice and Computers*. 1975.

[135] T. Heinbokel, S. Sonnentag, M. Frese, W. Stolte, and F. C. Brodbeck. Don't underestimate the problems of user centredness in software development projects - there are many! *Behaviour & Information Technology*, 15(4):226–236, 1996.

[136] J. Heiskari and L. Lehtola. Investigating the State of User Involvement in Practice. In *Proceedings of 16th Asia-Pacific Software Engineering Conference*, pages 433–440, Penang, Malaysia, 2009. IEEE.

[137] J. Herlocker, J. Konstan, L. Terveen, and J. Riedl. Evaluating Collaborative Filtering Recommender Systems. *ACM Transactions on Information Systems*, 1(22):5–53, 2004.

[138] K. Holtzblatt and S. Jones. Contextual inquiry: A participatory technique for system design. In D. Schuler and A. Namioka, editors, *Participatory design: Principles and practices*, pages 177–210. Erlbaum, Hillsdale, NJ, USA, 1993.

[139] Y. Hong, J. Lu, and J. Yao. What Reviews are Satisfactory: Novel Features for Automatic Helpfulness Voting. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 495–504, Portland, Oregon, USA, 2012. ACM.

[140] K.-Y. Huang. Challenges in human-computer interaction design for mobile devices. In *Proceedings of the World Congress on Engineering and Computer Science*, San Francisco, CA, USA, 2009.

[141] IEEE. IEEE Xplore. Last accessed: 28.08.2012, 2012. Available from: `http://ieeexplore.ieee.org/Xplore/dynhome.jsp`.

[142] IEEE Computer Society. IEEE Standard for Configuration Management in Systems and Software Engineering. Technical Report March, 2012.

[143] International Organization for Standardization. ISO 9241-11:1998. Ergonomics of Human System Interaction: Guidance on usability. Technical report, Geneva, Switzerland, 1998.

[144] International Organization for Standardization. ISO/IEC 14764. Software Engineering - Software Life Cycle Processes - Maintenance. Technical report, Geneva, Switzerland, 2006.

[145] International Organization for Standardization. ISO FDIS 9241-210:2009. Ergonomics of human system interaction - Part 210: Human-centered design for interactive systems. Technical report, Geneva, Switzerland, 2009.

[146] B. Ives and M. H. Olson. User Involvement and MIS Success: a Review of Research. *Management Science*, 30(5), 1984.

[147] A. Jameson, S. Baldes, and T. Kleinbauer. Two Methods for Enhancing Mutual Awareness in a Group Recommender System. In *Proceedings of the working conference on Advanced visual interfaces*, pages 447–449. ACM, 2004.

[148] A. Jameson and B. Smyth. Recommendation to Groups. In P. Brusilovsky, A. Kobsa, and W. Nejdl, editors, *The Adaptive Web, LNCS 4321*, pages 596–627. Springer-Verlag Berlin Heidelberg, 2007.

[149] R. Jameson. What is an Issue. Last accessed: 13.09.2012, 2012. Available from: `https://confluence.atlassian.com/display/JIRA/What+is+an+Issue`.

[150] S. Jobs. The Entrepreneur of the Decade Award, 1989. Available from: `http://www.inc.com/magazine/19890401/5602.html`.

[151] A. Jøsang. Trust-Based Decision Making for Electronic Transactions. In *Proceedings of the Fourth Nordic Workshop on Secure Computer Systems (NORDSEC'99)*, pages 1–21, Kista, Sweden, 1999.

[152] A. Jøsang. A logic for uncertain probabilities. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 9(3):279–311, 2001.

[153] A. Jøsang and R. Ismail. The beta reputation system. In *Proceedings of the 15th Bled Electronic Commerce Conference*, Bled, Slovenia, 2002.

[154] A. Jøsang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644, Mar. 2007.

[155] A. Kankainen. *Thinking model and tools for understanding user experience related to information appliance product concepts*. Dissertation, Helsinki University of Technology, 2002.

[156] A. Kanstrup and E. Christiansen. Selecting and evoking innovators: combining democracy and creativity. In *Proceedings of the 4th Nordic conference on Human-computer interaction*, pages 321–330. ACM, 2006.

[157] A. M. Kaplan and M. Haenlein. Users of the world, unite! The challenges and opportunities of Social Media. *Business Horizons*, 53(1):59–68, Jan. 2010.

[158] F. Kensing and K. H. Madsen. Generating visions: future workshops and metaphorical design. In J. Greenbaum and M. Kyng, editors, *Design at work: Cooperative design of computer systems*, pages 155–168. Erlbaum, Hillsdale, NJ, USA, Jan. 1992.

[159] F. Kensing and A. Munk-Madsen. PD: structure in the toolbox. *Communications of the ACM*, 36(6):78–85, June 1993.

[160] A. Kittur, E. H. Chi, and B. Suh. Crowdsourcing user studies with Mechanical Turk. In *Proceeding of the twenty-sixth annual CHI conference on Human factors in computing systems - CHI '08*, pages 453–456, Florence, Italy, 2008. ACM.

[161] A. Kjær and K. H. Madsen. Participatory analysis of flexibility. *Communications of the ACM*, 38(5):53–60, May 1995.

[162] A. J. Ko, M. J. Lee, V. Ferrari, S. Ip, and C. Tran. A case study of post-deployment user feedback triage. In *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering - CHASE '11*, pages 1–8, Honolulu, HI, USA, 2011. ACM.

[163] J. A. Konstan, J. Riedl, A. I. Borchers, and J. L. Herlocker. Recommender Systems: A GroupLens Perspective. In *Recommender Systems: Papers from the 1998 Workshop (AAAI Technical Report WS-98-08)*, pages 60–64, Menlo Park, CA, 1998. AAAI Press.

[164] J. Kontio, J. Bragge, and L. Lehtola. The Focus Group Method as an Empirical Tool in Software Engineering. In F. Shull, J. Singer, and D. I. K. Sjøberg, editors, *Guide to Advanced Empirical Software Engineering*, chapter 4. Springer-Verlag London, 2008.

[165] S. Kujala. User involvement: a review of the benefits and challenges. *Behaviour & information technology*, 22(1):1–16, 2003.

[166] S. Kujala and M. Kauppinen. Identifying and selecting users for user-centered design. In *Proceedings of the third Nordic conference on Human-computer interaction - NordiCHI '04*, pages 297–303, Tampere, Finland, 2004. ACM.

[167] S. Kujala, M. Kauppinen, L. Lehtola, and T. Kojo. The role of user involvement in requirements quality and project success. In *13th IEEE International Conference on Requirements Engineering (RE'05)*, pages 75–84, Paris, France, 2005. IEEE.

[168] D. Lafrenière. CUTA: a simple, practical, low-cost approach to task analysis. *interactions*, 3(5):35–39, Sept. 1996.

[169] A. Landini. Final Cut Pro X is Not a Professional Application, 2011. Available from: `http://www.petitiononline.com/finalcut/petition.html`.

[170] E. L.-C. Law, V. Roto, M. Hassenzahl, A. Vermeeren, and J. Kort. Understanding, Scoping and Defining User eXperience: A Survey Approach. In *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems*, pages 719–728, Boston, MA, USA, 2009. ACM.

[171] M. Lehman. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076, 1980.

[172] R. Levien. *Attack Resistant Trust Metrics.* Ph. d. thesis, UC Berkeley, 2004.

[173] B. Libert. *Social Nation: How to Harness the Power of Social Media to Attract Customers, Motivate Employees, and Grow Your Business.* Wiley, 2010.

[174] R. Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 140(140):1–55, 1932.

[175] V. Loftness. Improving Building Energy Efficiency in the U.S: Technologies and Policies for 2010 to 2050. In *Proceedings of the Pew Center/NCEP Workshop on the 10-50 Solution - Technologies and Policies for a Low-Carbon Future*, pages 60–68, Washington, DC, USA, 2004.

[176] S. Lohmann, S. Dietzold, P. Heim, and N. Heino. A Web Platform for Social Requirements Engineering. In *Proceedings of Software Engineering (Workshops)*, pages 309–315. GI, 2009.

[177] G. Longworth. *A User's Guide to SSADM Version 4.* NCC Blackwell, Oxford, 1992.

[178] J. Lowensohn. Petition seeks to bring back old Final Cut Pro. Last accessed: 08.12.2012, 2011. Available from: `http://news.cnet.com/8301-27076_3-20074841-248/petition-seeks-to-bring-back-old-final-cut-pro/`.

[179] W. Maalej. *Intention-Based Integration of Software Engineering Tools.* Dissertation, Technische Universität München, 2010.

[180] W. Maalej and H.-J. Happel. Can development work describe itself? In *7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pages 191–200, Cape Town, South Africa, May 2010. IEEE.

[181] W. Maalej, H.-J. H. Happel, and A. Rashid. When users become collaborators: towards continuous and context-aware user input. In *OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference companion on Object*

*oriented programming systems languages and applications*, pages 981–990, Orlando, FL, USA, 2009. ACM.

[182] W. Maalej and D. Pagano. On the Socialness of Software. In *Proceedings of the International Conference on Social Computing and its Applications*, Sydney, Australia, 2011. IEEE.

[183] W. Maalej, D. Panagiotou, and H.-J. Happel. Towards Effective Management of Software Knowledge Exploiting the Semantic Wiki Paradigm. In K. Herrmann and B. Brügge, editors, *Software Engineering*, pages 183–197, Bonn, Germany, 2008. GI.

[184] L. Macaulay. Cooperation in understanding user needs and requirements. *Computer Integrated Manufacturing Systems*, 8(2):155–165, May 1995.

[185] N. Machado, P. Romano, and L. Rodrigues. Lightweight Cooperative Logging for Fault Replication in Concurrent Programs. In *Proceedings of the 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, Boston, MA, USA, 2012. IEEE.

[186] A. MacLean, K. Carter, L. Lövstrand, and T. Moran. User-tailorable systems: pressing the issues with buttons. In *Proceedings of the SIGCHI conference on Human factors in computing systems Empowering people - CHI '90*, pages 175–182, Seattle, WA, USA, Mar. 1990. ACM.

[187] K. H. Madsen and P. H. Aiken. Experiences using cooperative interactive storyboard prototyping. *Communications of the ACM*, 36(6):57–64, June 1993.

[188] U. Malinowski and K. Nakakoji. Using computational critics to facilitate long-term collaboration in user interface design. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '95*, pages 385–392, Denver, CO, USA, May 1995. ACM.

[189] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann. Design Lessons from the Fastest Q&A Site in the West. In *Proceedings of the 2011 annual conference on Human factors in computing systems - CHI '11*, pages 2857–2866, Vancouver, BC, Canada, May 2011. ACM.

[190] D. W. Manchala. Trust Metrics, Models and Protocols for Electronic Commerce Transactions. In *Proceedings of the 18th International Conference on Distributed Computing Systems*, pages 312–321, Amsterdam, Netherlands, 1998. IEEE.

[191] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval.* Cambridge University Press, 2008.

[192] P. Y. Martin and B. A. Turner. Grounded Theory and Organizational Research. *The Journal of Applied Behavioral Science*, 22(2):141–157, Apr. 1986.

[193] K. McCarthy, M. Salamó, L. Coyle, L. McGinty, B. Smyth, and P. Nixon. Group recommender systems: a critiquing based approach. In *Proceedings of the 11th international conference on Intelligent user interfaces*, pages 267–269, Sydney, Australia, 2006. ACM.

[194] A. McFarland and T. Dayton. A participatory methodology for driving object-oriented GUI design from user needs. In *Proceedings of OZCHI95, the CHISIG Annual Conference on Human-Computer Interaction*, Wollongong, New South Wales, Australia, 1995.

[195] F. C. Mish, editor. *Merriam-Webster's Collegiate Dictionary.* Merriam-Webster, USA, 11th edition, 2003.

[196] L. Mui, M. Mohtashemi, and A. Halberstadt. A computational model of trust and reputation. In *Proceedings of the 35th Hawaii International Conference on System Sciences*, pages 1–9, Waikoloa, HI, USA, 2002. IEEE.

[197] M. J. Muller. PICTIVE - an exploration in participatory design. In *Proceedings of the SIGCHI conference on Human factors in computing systems Reaching through technology - CHI '91*, pages 225–231, New Orleans, LA, USA, Mar. 1991. ACM.

[198] M. J. Muller, J. Hallewell Haslwanter, and T. Dayton. Participatory Practices in the Software Lifecycle. In M. Helander, T. K. Landauer, and P. Prabh, editors, *Handbook of Human-Computer Interaction*, chapter 11, pages 255–297. Elsevier Science B.V, 2nd edition, 1997.

[199] M. J. Muller, A. McLard, B. Bell, S. Dooley, L. Meiskey, J. A. Meskill, R. Sparks, and D. Tellam. Validating an Extension to Participatory Heuristic Evaluation: Quality of Work and Quality of Work Life. In *Proceedings of CHI '95 Conference Companion on Human Factors in Computing Systems*, pages 115–116, Denver, CO, USA, 1995. ACM.

[200] M. J. Muller, L. G. Tudor, D. M. Wildman, E. A. White, R. W. Root, T. Dayton, R. Carr, B. Diekmann, and E. Dykstra-Erickson. Bifocal tools for scenarios and representations in participatory activities with users. In J. Carroll, editor, *Scenario-based design*, pages 135–163. John Wiley & Sons, Inc. New York, NY, USA, Sept. 1995.

[201] M. J. Muller, D. M. Wildman, and E. A. White. Participatory design through games and other group exercises. In *Conference companion on*

*Human factors in computing systems - CHI '94*, pages 411–412, Boston, MA, USA, Apr. 1994. ACM.

[202] E. Mumford. Consensus Systems Design: An Evaluation of this Approach. In N. Szyperski and E. Grochela, editors, *Design and Implementation of Computer-Based Information Systems*. Sijthoff & Noordhoff, Groningen, Netherlands, 1979.

[203] E. Mumford and M. Weir. *Computer systems in work design–The ETHICS method: Effective technical and human implementation of computer systems*. Wiley Press, New York, USA, 1979.

[204] S. Narayanasamy, G. Pokam, and B. Calder. BugNet: Continuously Recording Program Execution for Deterministic Replay Debugging. In *32nd International Symposium on Computer Architecture (ISCA'05)*, pages 284–295, Madison, WI, USA, 2005. IEEE.

[205] J. Nielsen. *Usability Engineering*. Morgan Kaufmann, San Francisco, CA, USA, 1993.

[206] J. Nielsen. The Use and Misuse of Focus Groups. *IEEE Software*, 14(1):94–95, 1997.

[207] J. Nielsen. First Rule of Usability? Don't Listen to Users. Last accessed: 08.12.2012, 2001. Available from: `http://www.useit.com/alertbox/20010805.html`.

[208] D. A. Norman. *The Design of Everyday things*. MIT Press, London, England, 2002.

[209] D. A. Norman and S. W. Draper. *User Centered System Design; New Perspectives on Human-Computer Interaction*. Erlbaum, 1986.

[210] D. A. Norman, J. Miller, and A. Henderson. What You See, Some of What's in the Future, And How We Go About Doing It: HI at Apple Computer. In *CHI '95 Conference companion on Human factors in computing systems*, page 155, Denver, Colorado, USA, 1995. ACM.

[211] K. Noro and A. S. Imada, editors. *Participatory ergonomics*. Taylor and Francis, London, England, 1991.

[212] B. Nuseibeh and S. Easterbrook. Requirements Engineering: A Roadmap. In *Conference on the Future of Software Engineering*, pages 35–46, Limerick, Ireland, 2000. ACM.

[213] M. O'Connor, D. Cosley, J. Konstan, and J. Riedl. PolyLens: A recommender system for groups of users. In *European Conference on Computer-Supported Cooperative Work*, pages 199–218, 2001.

[214] S. Olejnik and J. Algina. Measures of Effect Size for Comparative Studies: Applications, Interpretations, and Limitations. *Contemporary educational psychology*, 25(3):241–286, July 2000.

[215] T. O'Reilly. What Is Web 2.0, 2005. Available from: `http://oreilly.com/web2/archive/what-is-web-20.html?page=4`.

[216] D. Özçelik Buskermolen, J. Terken, and B. Eggen. Informing User Experience Design About Users: Insights from Practice. In *Proceedings of the 2012 ACM annual conference extended abstracts on Human Factors in Computing Systems Extended Abstracts (CHI EA'12)*, pages 1757–1762, Austin, Texas, USA, 2012. ACM.

[217] D. Pagano. Towards Systematic Analysis of Continuous User Input. In *Proceedings of the 4th International Workshop on Social Software Engineering*, pages 6–10, Szeged, Hungary, 2011. ACM.

[218] D. Pagano and B. Bruegge. User Involvement in Software Evolution Practice: A Case Study. In *Proceedings of the 35th International Conference on Software Engineering*, San Francisco, CA, USA, 2013. IEEE.

[219] D. Pagano, M. A. Juan, A. Bagnato, T. Roehm, B. Bruegge, and W. Maalej. FastFix: Monitoring Control for Remote Software Maintenance. In *Proceedings of the 34th International Conference of Software Engineering*, pages 1437–1438, Zurich, Switzerland, 2012. IEEE.

[220] D. Pagano and W. Maalej. How Do Developers Blog? An Exploratory Study. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 123–132, Honolulu, HI, USA, 2011. ACM.

[221] D. Pagano and W. Maalej. How Do Open Source Communities Blog? *International Journal on Empirical Software Engineering*, (May), 2012.

[222] D. Pagano and W. Maalej. User Feedback in the AppStore: An Empirical Study. In *Proceedings of the 21st International Requirements Engineering Conference*, Rio de Janeiro, Brasil, 2013. IEEE.

[223] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford InfoLab, 1999. Available from: `http://ilpubs.stanford.edu:8090/422`.

[224] Ø. Pålshaugen. Method of designing a starting conference. Technical report, Work Research Institute, Oslo, 1986.

[225] S. Park, Y. Zhou, W. Xiong, Z. Yin, R. Kaushik, K. H. Lee, and S. Lu. Pres: probabilistic replay with execution sketching on multiprocessors. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 177–192, Big Sky, MT, USA, 2009. ACM, ACM.

[226] C. Parnin and C. Treude. Measuring API documentation on the web. In *Proceedings of the 2nd International Workshop on Web 2.0 for Software Engineering*, pages 25–30, Honolulu, HI, USA, 2011. ACM.

[227] C. Parnin, C. Treude, L. Grammel, and M.-A. Storey. Crowd Documentation: Exploring the Coverage and the Dynamics of API Discussions on Stack Overflow. Technical report, Georgia Institute of Technology, 2012.

[228] F. Paternò, A. Piruzza, and C. Santoro. Remote Web usability evaluation exploiting multimodal information on user behavior. In G. Calvary, C. Pribeanu, G. Santucci, and J. Vanderdonckt, editors, *Computer-Aided Design of User Interfaces V*, pages 287–298. Springer Netherlands, 2007.

[229] M. Pazzani and D. Billsus. Learning and Revising User Profiles: The Identification of Interesting Web Sites. *Machine Learning*, 27:313–331, 1997.

[230] M. J. Pazzani and D. Billsus. Content-Based Recommendation Systems. In P. Brusilovsky, A. Kobsa, and W. Nejdl, editors, *The Adaptive Web, LNCS 4321*, pages 325–341. Springer-Verlag Berlin Heidelberg, 2007.

[231] S. M. Peters. *A Framework for the Intuitive Control of Smart Home and Office Environments*. Master's thesis, Technische Universität München, 2011.

[232] K. Pohl. *Process-Centered Requirements Engineering*. John Wiley and Sons, 1996.

[233] K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Sept. 2005.

[234] A. Porterfield, P. Khare, and A. Vahl. *Facebook Marketing All-in-One for Dummies*. John Wiley and Sons, 2011.

[235] A. Rashid. OpenProposal: Towards Collaborative End-User Participation in Requirements Management By Usage of Visual Requirement Specifications. In *15th IEEE International Requirements Engineering Conference (RE 2007)*, pages 371–374, New Delhi, India, 2007. IEEE.

[236] A. Rashid, J. Wiesenberger, D. Meder, and J. Baumann. Bringing developers and users closer together: The OpenProposal story. In *Multikonferenz Wirtschaftsinformatik*, 2008.

[237] M. Reinhardt, G. Groh, and M. Wiener. Web 2.0 driven Open Innovation Networks-A Social Network Approach to Support the Innovation Context within Companies. In *MKWI 2010 - E-Commerce and E-Business*, pages 1177–1190, Göttingen, Germany, 2010. Universitätsverlag Göttingen.

[238] P. Resnick. Trust among strangers in Internet transactions: Empirical analysis of eBay's reputation system. *Advances in Applied Microeconomics*, 11(The Economics of the Internet and E-Commerce):1–26, 2002.

[239] F. Ricci, L. Rokach, B. Shapira, and P. Kantor, editors. *Recommender Systems Handbook*. Springer Verlag, 1st edition, 2011.

[240] W. M. Riggs and E. von Hippel. The Impact of Scientific and Commercial Values on the Sources of Scientific Instrument Innovation. *Research Policy*, 23:459–469, 1992.

[241] M. P. Robillard, R. J. Walker, and T. Zimmermann. Recommendation Systems for Software Engineering. *IEEE Software*, 27(4):80–86, 2010.

[242] C. Robson. *Real World Research*. Wiley & Sons, 3rd edition, 2011.

[243] J. L. Rodgers and W. A. Nicewander. Thirteen Ways to Look at the Correlation Coefficient. *The American Statistician*, 42(1):59–66, 1988.

[244] T. Roehm, R. Tiarks, R. Koschke, and W. Maalej. How do professional developers comprehend software? In *2012 34th International Conference on Software Engineering (ICSE)*, pages 255–265, Zurich, Switzerland, 2012. IEEE.

[245] S. Rosenbaum, J. A. Rohn, and J. Humburg. A Toolkit for Strategic Usability: Results from Workshops, Panels, and Surveys. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 337–344, The Hague, Netherlands, 2000. ACM.

[246] R. Rothwell, C. Freeman, A. Horlsey, V. T. P. Jervis, A. B. Robertson, and J. Townsend. SAPPHO updated - project SAPPHO phase II. *Research Policy*, 3(3):258–291, 1974.

[247] V. Roto, E. Law, A. Vermeeren, and J. Hoonhout, editors. *User Experience White Paper*. 2011. Available from: `http://www.allaboutux.org/files/UX-WhitePaper.pdf`.

[248] V. Roto, M. Lee, K. Pihkala, B. Castro, A. Vermeeren, E. Law, K. Väänänen-Vainio-Mattila, J. Hoonhout, and M. Obrist. User experience definitions. Last accessed: 22.07.2012, 2012. Available from: `http://www.allaboutux.org/ux-definitions`.

[249] W. W. Royce. Managing the development of large software systems. *Proceedings of IEEE WESCON 26*, (August):1–9, 1970.

[250] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, Dec. 2008.

[251] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[252] J. Sabater and C. Sierra. REGRET: A reputation model for gregarious societies. In C. Castelfranchi and L. Johnson, editors, *Proceedings of the 4th Int. Workshop on Deception, Fraud and Trust in Agent Societies*, volume 73, pages 194–195, Montreal, Canada, 2001. ACM.

[253] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc, New York, NY, USA, Oct. 1986.

[254] E. B. N. Sanders. Participatory design research in the product development process. In *PDC '92: Proceedings of the Participatory Design Conference*, pages 111–112, Cambridge, MA, USA, 1992.

[255] R. J. Sandusky. Software Problem Management as Information Management in a F/OSS Development Community. In *Proceedings of the First International Conference on Open Source Systems*, pages 44–49, Genova, Italy, 2005. IEEE.

[256] B. Schilit, N. Adams, and R. Want. Context-Aware Computing Applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, pages 89–101, Santa Cruz, CA, US, 1994. IEEE.

[257] A. Schmidt, M. Beigl, and H.-W. Gellersen. There is more to context than location. *Computers & Graphics*, 23(6):893–901, Dec. 1999.

[258] J. Schneider, G. Kortuem, J. Jager, S. Fickas, and Z. Segall. Disseminating trust information in wearable communities. *Personal Technologies*, 4(4):245–248, Dec. 2000.

[259] K. Schneider. Focusing spontaneous feedback to support system evolution. In *19th International Requirements Engineering Conference*, pages 165–174, Trento, Italy, Aug. 2011. IEEE.

[260] K. Schneider, S. Meyer, M. Peters, F. Schliephacke, J. Mörschbach, and L. Aguirre. Feedback in Context: Supporting the Evolution of IT-Ecosystems. In M. Ali Babar, M. Vierimaa, and M. Oivo, editors, *Product-Focused Software Process Improvement, LNCS 6156*, pages 191–205. Springer-Verlag Berlin Heidelberg, 2010.

[261] K. Schwaber. SCRUM Development Process. In *Business Object Design and Implementation Workshop, OOPSLA'95*, pages 10–19, Austin, Texas, USA, 1995.

[262] N. Seyff, F. Graf, and N. Maiden. Using Mobile RE Tools to Give End-Users Their Own Voice. In *IEEE International Conference on Requirements Engineering*, pages 37–46, Sydney, Australia, 2010. IEEE.

[263] H. Sharp, A. Finkelstein, and G. Galal. Stakeholder identification in the requirements engineering process. In *Proceedings of the 10th International Workshop on Database and Expert Systems Applications, DEXA 99*, pages 387–391, Florence, Italy, 1999. IEEE.

[264] L. Singer and K. Schneider. It was a Bit of a Race: Gamification of Version Control. In *Proceedings of the 2nd international workshop on Games and software engineering (GAS)*, pages 5–8, Zurich, Switzerland, June 2012. IEEE.

[265] R. Sinha. Persona development for information-rich domains. In *CHI '03 extended abstracts on Human factors in computer systems - CHI '03*, page 830, Fort Lauderdale, FL, USA, 2003. ACM.

[266] Springer. SpringerLink. Last accessed: 28.08.2012, 2012. Available from: `http://link.springer.com`.

[267] S. M. Srinivasan, S. Kandula, C. R. Andrews, and Y. Zhou. Flashback: a lightweight extension for rollback and deterministic replay for software debugging. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 3–17, Boston, MA, USA, June 2004.

[268] H. Stangl. *Script : A Framework for Scenario-Driven Prototyping.* Dissertation, Technische Universität München, 2012.

[269] H. Stangl and O. Creighton. Continuous demonstration. In *Fourth International Workshop on Multimedia and Enjoyable Requirements Engineering (MERE'11)*, pages 38–41, Trento, Italy, 2011. IEEE.

[270] N. A. Streitz, J. Siegel, V. Hartkopf, and S. Konomi, editors. *Cooperative Buildings: Integrating Information, Organizations, and Architecture.* Springer, 1999.

[271] TestFlight App Inc. TestFlight. Last accessed: 09.11.2012, 2012. Available from: `https://testflightapp.com`.

[272] The Nielsen Company. Led by Facebook, Twitter, Global Time Spent on Social Media Sites up 82% Year over Year. Last accessed: 09.11.2012, 2010. Available from: `http://blog.nielsen.com/nielsenwire/global/led-by-facebook-twitter-global-time-spent-on-social-media-sites-up-82-year-over-year/`.

[273] C. Treude and M.-A. Storey. How tagging helps bridge the gap between social and technical aspects in software development. In *ICSE '09: Proceedings of the 2009 IEEE 31st International Conference on Software Engineering*, pages 12–22, Washington, DC, USA, 2009. IEEE Computer Society.

[274] M. W. Tschudy, E. A. Dykstra-Erickson, and M. S. Holloway. Picture-CARD : A Storytelling Tool for Task Analysis. In *PDC'96 Proceedings of the Participatory Design Conference.*, pages 183–191, Cambridge, MA, USA, 1996.

[275] I. Tuomi. *Networks of Innovation: Change and Meaning in the Age of the Internet.* Oxford University Press, USA, 2006.

[276] URCOT. Work mapping: Possible application in the Australia Taxation Office. Technical report, Union Research Centre on Organisation and Technology Ltd. (URCOT), Melbourne, Australia, 1994.

[277] UserVoice Inc. Feedback and Online Help Desk Software. Last accessed: 08.04.2012. Available from: `http://www.uservoice.com`.

[278] A. van Deursen, A. Mesbah, B. Cornelissen, A. Zaidman, M. Pinzger, and A. Guzzi. Adinda: A Knowledgeable, Browser-Based IDE. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, pages 203–206, Cape Town, South Africa, 2010. ACM.

[279] A. Vargas, H. Weffers, and H. Viera de Rocha. A method for remote and semi-automatic usability evaluation of web-based applications through users behavior analysis. In *Proceedings of the 7th International Conference on Methods and Techniques in Behavioral Research, MB'10*, Eindhoven, Netherlands, 2010. ACM.

[280] D. Vergados, A. Alevizos, A. Mariolis, and M. Caragiozidis. Intelligent services for assisting independent living of elderly people at home. In *Proceedings of the 1st ACM international conference on PErvasive Technologies Related to Assistive Environments - PETRA '08*, pages 1–4, Athens, Greece, July 2008. ACM.

[281] E. von Hippel. Lead Users: A Source of Novel Product Concepts. *Management Science*, 32(7):791–805, July 1986.

[282] E. von Hippel. Innovation by User Communities: Learning from Open-Source Software. *MIT Sloan Management Review*, Summer, 2001.

[283] E. von Hippel. *Democratizing Innovation.* MIT Press, 2005.

[284] K. Vredenburg, J.-Y. Mao, P. W. Smith, and T. Carey. A survey of user-centered design practice. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '02*, pages 471–478, Minneapolis, Minnesota, USA, 2002. ACM.

[285] J. West and S. Gallagher. Open Innovation : The Paradox of Firm Investment in Open Source Software. *R&D Management*, 36(3), 2004.

[286] A. Whitby, A. Jøsang, and J. Indulska. Filtering out unfair ratings in bayesian reputation systems. *Science*, 4(2)(2):106–117, 2004.

[287] M. G. Williams and V. Begg. Translation between software designers and users. *Communications of the ACM*, 36(6):102–103, June 1993.

[288] S. Wilson, M. Bekker, P. Johnson, and H. Johnson. Helping and Hindering User Involvement - A Tale of Everyday Design. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, Atlanta, Georgia, 1997. ACM.

[289] D. Wixon, K. Holtzblatt, and S. Knox. Contextual design: an emergent view of system design. In *Proceedings of the SIGCHI conference on Human factors in computing systems Empowering people - CHI '90*, pages 329–336, Seattle, WA, USA, Mar. 1990. ACM.

[290] D. R. Wixon, C. M. Pietras, P. K. Huntwork, and D. W. Muzzey. Changing the rules: a pragmatic approach to product development. In *Field methods casebook for software design*, pages 57–89. John Wiley & Sons, Inc. New York, NY, USA, Sept. 1996.

[291] J. Wood and D. Silver. *Joint Application Design.* Wiley Press, New York, USA, 1989.

[292] P. C. Wright and A. F. Monk. A cost-effective evaluation method for use by designers. *International Journal of Man-Machine Studies*, 35(6):891–912, Dec. 1991.

[293] T. Yamakami. A Three-Dimension Analysis of Driving Factors for Mobile Application Stores: Implications of Open Mobile Business Engineering.

In *2011 IEEE Workshop of International Conference on Advanced Information Networking and Applications*, pages 885–889, Biopolis, Singapore, 2011. IEEE.

[294] E. Young and R. Greenlee. Participatory video prototyping. In *Posters and short talks of the 1992 SIGCHI conference on Human factors in computing systems - CHI '92*, pages 28–28, Monterey, CA, USA, May 1992. ACM.

[295] M. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390, 2000.

[296] W. Zhou, Y. Zhou, X. Jiang, and P. Ning. Detecting Repackaged Smartphone Applications in Third-Party Android Marketplaces. In *Proceedings of the second ACM conference on Data and Application Security and Privacy*, pages 317–326, San Antonio, Texas, USA, 2012. ACM.

[297] C. N. Ziegler and G. Lausen. Spreading Activation Models for Trust Propagation. In *Proceedings of the IEEE International Conference on e-Technology, e-Commerce, and e-Service (EEE '04)*, Taipei, Taiwan, 2004.

[298] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schröter, and C. Weiss. What Makes a Good Bug Report? *IEEE Transactions on Software Engineering*, 36(5):618–643, Sept. 2010.