User Involvement in Software Evolution Practice: A Case Study

Dennis Pagano
Technische Universität München
Munich, Germany
pagano@cs.tum.edu

Bernd Bruegge
Technische Universität München
Munich, Germany
bruegge@cs.tum.edu

***Abstract*—User involvement in software engineering has been researched over the last three decades. However, existing studies concentrate mainly on early phases of user-centered design projects, while little is known about how professionals work with post-deployment end-user feedback. In this paper we report on an empirical case study that explores the current practice of user involvement during software evolution.**

We found that user feedback contains important information for developers, 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, which is mostly accomplished manually and consequently requires high effort. Overall, our results show the need for tool support to consolidate, structure, analyze, and track user feedback, particularly when feedback volume is high. Our findings call for a hypothesis-driven analysis of user feedback to establish the foundations for future user feedback tools.

***Index Terms*—user involvement, software evolution, user feedback**

I. Introduction

User involvement in software engineering is an established research field [10], which has been studied particularly at its intersection with human-computer-interaction [1]. It aims at maximizing system usefulness and usability by understanding users' needs and expectations. Its foundations go back to the early 1980s, in particular to Gould's and Lewis' key principles of usability [4] which together with Norman's and Draper's work [12] form the pillars of user-centered design.

Over the last three decades user involvement has changed significantly. For example, software users changed from programmers or trained technical staff to practically any person [5], causing a fundamental shift in developers' attitude towards them [8]. With application distribution platforms and mobile devices, neither the users of software nor its context of use are known before its delivery. The consequence of these changes is an increasing distance between developers and users [6], while more focus on users would actually be necessary to satisfy their increasing demands [11]. As a consequence, post-deployment user feedback such as feature requests and bug reports become increasingly important to developers [9].

In open source communities social media facilitate the interaction between users and developers and help to dissolve boundaries between them [14]. Likewise, current web-based or integrated rating and feedback platforms allow users to easily express their pains and needs for commercial software. But while there are studies about user involvement in user-centered design projects, little is known about how professionals work with end-user feedback, and which are the benefits and challenges – especially when a large number of users is involved.

We report on an empirical case study which we conducted between June and September 2012 to explore the current practice of user involvement during software evolution. Our goal was to understand what happens with user feedback in development environments and what is the rationale behind. First and foremost, we aimed at describing current practice and identifying underlying challenges, but we also wanted to glimpse at the practitioners' requirements on tools which would solve the identified issues. The contribution of this paper is twofold. First, we describe settings put in place by professionals to involve users and how and why they deal with the collected user feedback. Second, we derive a catalogue of hypotheses about user involvement in software evolution to guide further research and tool development efforts.

The paper is organized as follows. Section II explains the study setting in terms of research questions and methodology, and introduces the research data. The following three sections summarize our findings on the user involvement setting (Section III), developers' workflows to analyze user feedback (Section IV), and developers' expectations towards tool support (Section V). Section VI discusses implications of our findings for researchers, practitioners, and tool designers and describes the limitations of our study, while Section VII presents related work. Section VIII concludes our study.

II. Study Setting

A. Research Questions

We are particularly interested in three aspects of how professional developers involve users during software evolution: 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 1: User involvement setting describes *how*, *where*, and *when* users may provide feedback. In particular, we aim at answering the following questions:

- *Infrastructure*: Which channels do developers use to gather user feedback?
- *Frequency*: How often can users provide feedback?

978-1-4673-3076-3/13/$31.00 © 2013 IEEE

ICSE 2013, San Francisco, CA, USA
Software Engineering in Practice

- *Communication*: Are users systematically involved?

RQ 2: 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 3: User involvement requirements summarize developers' *needs* and *expectations* towards tool support for user involvement. We focus particularly on user feedback consolidation and assessment, answering the following questions:

- *Tool support*: Do developers expect that tool support for feedback consolidation improves user involvement practice?
- *Consolidation*: How should feedback be consolidated?
- *Assessment*: Would developers embrace measures about users and usage which are driven 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 about the subjects' background and the projects they report on, in order to enable a descriptive classification. The complete catalogue of questions is available online[1].

B. Research Method

Case studies can provide a detailed view on real-life situations and thus may reveal the reasons why and the mechanisms how specific phenomena occur [2], [3]. Our goal is to explore user involvement in software evolution practice to understand developers' current workflows, their problems, and their motivation to involve users. Additionally, we seek to describe and explain the situation whenever possible. Consequently, our case study mainly serves an *exploratory* purpose, but also exposes *descriptive* or *explanatory* character [16].

Our *study proposition* is threefold. *First*, we assume that developers appreciate user feedback as important source of information about the acceptance of their software. *Second*, we hypothesize 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 [13]. *Third*, we expect 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 [15] via phone or face-to-face, with professional developers working at different software companies. Semi-structured interviews allow for improvisation and thus facilitate an exploration of the studied cases [16]. Rather than randomly choosing cases for the study, 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 employ this *purposive* sampling strategy since we expect the resulting cases to be most relevant to our proposition [2].

[1] http://www1.cs.tum.edu/dennispagano#downloads

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 1. Case study methodology.

For our case study, we followed the design principles and procedures described by Runeson and Höst [16]. As shown in Figure 1, our methodology consists of three phases: a preparation phase, a data collection phase, and a data analysis phase.

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 via email to six selected companies which were the first candidates. Five companies 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.

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 [16], 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

Table I
OVERVIEW OF INTERVIEW PARTNERS AND STUDIED PROJECTS.

company code	# active users	user audience	platform	release cycle	subject code	work experience	project role
C1	> 10,000	special consumers	both	3 weeks	S1	6–10 years	developer
C2	150,000–200,000	consumers & professionals	desktop	4 weeks	S2	3–5 years	developer
C3	> 10,000	consumers	mobile	2–6 months	S3	3–5 years	developer
C4	100,000–200,000	consumers	mobile	2–6 months	S4	3–5 years	architect
C5	100–500	special professionals	desktop	4–8 weeks	S5	6–10 years	product manager

each transcription to the corresponding interviewee and asked for corrections, which we did not receive.

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 questions. Moreover, subjects often extended earlier statements as the interview proceeded, so that helpful details were frequently given during later questions. We therefore decided to code the transcriptions according to two schemata. First, the pre-defined interview question each statement answers, 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.

C. Research Data

Five small and medium-sized software companies agreed to take part in our case study. We interviewed one subject per company. Each subject reported on one specific, proprietary, commercially successful product developed in her company. Table I gives an overview of the studied projects and subjects. In particular, it provides details about the user audience, and shows the work experience and main project role for each subject. Four of the five studied software products (C1–C4) have a large number of users (more than 10,000), whereas the software developed by C5 accounts for a smaller user audience (between 100 and 500). In three cases (C1, C3, and C4) the users of the studied software are *consumers*, a *special consumer group* in the case of C1, and *all consumers* in the case of C3 and C4. From the remaining two companies, C2 develops an application with a *general* user audience, while C5 develops software for a *special group of professionals*. Two companies (C2 and C5) work on *desktop* applications and two (C3 and C4) on *mobile* applications, while in the case of C1 the application is developed for *both* platforms.

III. USER INVOLVEMENT SETTING

Our interviews show that gathering user feedback plays an important role in the studied companies. 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, except S5, also regard users' *ratings* (e.g. in application distribution platforms such as the Apple AppStore) 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*, which suggests that public ratings are less important in such contexts. In contrast, S1, who develops 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".

Table II
USER INVOLVEMENT INFRASTRUCTURE: USER FEEDBACK CHANNEL UTILIZATION FREQUENCY – 4: MOST OFTEN, 3: OFTEN, 2: SOMETIMES, 1: RARELY, –: NEVER.

	channel	C1	C2	C3	C4	C5	mean
Error Reports	Integrated	3	4	4	3	4	3.6
	Email	4	1	4	1	4	2.8
	AppStore	–	–	4	4	–	1.6
	Phone	–	1	3	1	1	1.2
	Blog	–	3	–	–	–	0.6
Feature Requests	Email	3	2	4	1	4	2.8
	AppStore	–	3	4	4	–	2.2
	Integrated	–	4	3	3	–	2.0
	Phone	–	2	–	1	1	0.8
	Blog	–	3	–	–	–	0.6
	Forum	3	–	–	–	–	0.6
	Twitter	–	3	–	–	–	0.6
	Facebook	1	–	–	–	–	0.2
Feedback	Email	3	2	4	1	4	2.8
	AppStore	3	2	4	4	–	2.6
	Integrated	–	4	–	3	–	1.4
	Phone	–	–	3	1	1	1.0
	Blog	–	3	–	–	–	0.6
	Forum	3	–	–	–	–	0.6
Ratings	AppStore	4	–	4	4	–	2.4
	Rating site	–	4	–	–	–	0.8
	Email	–	–	3	–	–	0.6

A. Infrastructure

Table II 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. 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. S3 specifically 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 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 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.*

B. Frequency

In all companies we studied, users can provide feedback continuously. In practice this means that the feedback channels remain continuously open. Companies C1, C2, and C4 receive a continuous stream of user feedback, with dozens of messages per day, even if no new version was recently released. C3 and C5 receive less feedback, each for a different reason. C3 develops software on behalf of other companies. Therefore, they typically get feedback digests, pre-selected user feedback, or feedback reported by their customers. The users of C5, who all use the software professionally, first collect several ideas and feedback and then send a 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."

Table III
USER-DEVELOPER COMMUNICATION MODES.

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

Hypothesis 3. *Users frequently provide feedback, but user feedback does not always reach developers.*

C. 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 as 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 III. *Error reports* are automatically triggered (pull) in the case of C1, C2, and C5, who ask users to provide 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 roll out beta versions to selected users. Finally, C4 explicitly trigger product *ratings* after fixed time periods by redirecting the user to a specific AppStore page.

An interesting observation was made by S5, who distinguished four different communication types. First, conversations with users at trade 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 4. *Users are not systematically involved during software evolution. Apart from error reports there is no commonly agreed practice on how to* provide *nor on how to* gather *user feedback during software evolution.*

IV. User Involvement Workflow

A. 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, the user is king. After all, users buy the product and thus companies are interested to satisfy their needs. 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 5. *User feedback supports 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.

a) Improve Quality: Users typically help to improve software quality by reporting latent errors. S4 explains that error 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 to be significant instead, which eventually affected ratings and sales numbers.

Hypothesis 6. *User feedback can improve software quality.*

b) Identify Missing Features: All subjects reported that users 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

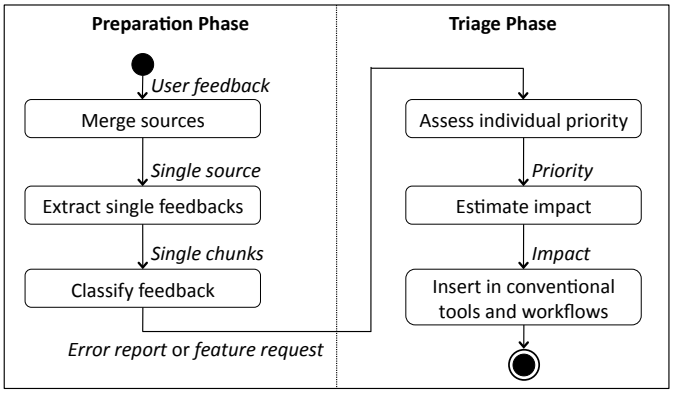

Figure 2. User feedback analysis process.

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 how they could improve their workflows and increase their profit. Then they argue that it would help other users as well."

Hypothesis 7. *User feedback helps to identify missing features, but developers need to assess how many users will benefit from a specific new feature.*

c) Advertise and Market Product: Our 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 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 on 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 8. *User feedback helps to advertise and market a product by conveying trust in the form of positive user ratings and user experience.*

B. Analysis

Our 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 2.

d) 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.

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

Hypothesis 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.*

C. 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 how many users it applies to. We identified three main reasons: content and quality of user feedback, manual analysis and quantity of feedback, and the communication gap between developers and users.

f) 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, complex features and user workflows often lead to complex feature requests or issues. All subjects except 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 provide any value to the developers. However, developers can only estimate the value of feedback after reading or at least browsing it. It seems that such wasted time 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."

Table IV
PERCEIVED COMPLEXITY OF AND SATISFACTION WITH CURRENT USER FEEDBACK ANALYSIS PRACTICE.

subject	amount of feedback	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

Hypothesis 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.*

g) Manual Analysis and Quantity of Feedback: We found that 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*, 2: *unsatisfied* 3: *undecided*, 4: *satisfied*, 5: *very satisfied*). The average response lies between *somewhat unsatisfied* and *undecided* (mean=2.6), as shown in Table IV. 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 influence developers' satisfaction with current practice.

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

Hypothesis 11. *Developers analyze user feedback mainly manually and read single feedback multiple times.*

Hypothesis 12. *Developers spend most effort on assessing the priority and impact of user feedback. The main reason for this is that developers need to manually estimate how many users are affected by a specific feedback.*

h) 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 effort, for instance to ask clarification questions. Furthermore, it increases developers' distance to the reporting user. 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 the feedback author 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 13. *Users and developers are disconnected due to communication gaps in user feedback channels.*

V. USER INVOLVEMENT REQUIREMENTS

A. 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 an impact analysis. S4 pointed out that user feedback should be regarded as an 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 a small amount of feedback. Further, S1 and S5 remarked that new tools really need to provide significant benefits with respect to their current practices to be acceptable.

Hypothesis 14. *Developers need tool support to consolidate, structure, analyze, and track user feedback, particularly when feedback volume is high.*

B. 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 the reporting users relevant existing feedback that 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 developers' shoulders, since priority and impact are basically results of the number of votes. Second, it reduces or even 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 two different privacy scenarios: The re-active approach is more appropriate for feedback which is currently private to the company such as phone calls, while the pro-active approach should be employed for publicly visible feedback such as AppStore reviews.

When asked how tools 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* (error reports, feature requests, etc.) 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 15. *To consolidate feedback, tools should group duplicate or similar feedback, capture the feedback type, and the feature for which it applies.*

Hypothesis 16. *Pro-active tools are appropriate for high volumes of non-confidential feedback, while re-active tools are appropriate for less feedback and more professional end-users.*

C. 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 specific feedback, i.e. how many users provided the same or 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 upset our users." The main factors slowing down reaction time include low visibility of existing feedback for other users, scattering of feedback across multiple channels, as well as 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 reporting user has used the application, as pointed out 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 17. *To support impact analysis, tools should measure the frequency of user feedback and provide developers with an individual assessment of the reporting user.*

VI. Discussion

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

A. 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 effort is required for this analysis, as developers mostly accomplish it manually.

1) Implications for Researchers: In order to facilitate user feedback analysis, researchers should study 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 explore 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.

2) Implications for Practitioners: Based on our results, we give three recommendations 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 to their users how to provide helpful feedback. For instance, multiple requests within one message might complicate its analysis, while indicating the feedback type might be helpful.

3) Implications for Tool Designers: User feedback is a rich source of information. Our study shows 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 reporting user, for instance how often and for how long she has used the software or how often she has already reported.

B. Limitations

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

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 occurring problems. To limit this effect, we concretized questions related to problems, asking which tasks take much time, are subjectively difficult, and which information is needed to accomplish them. Other studies (e.g. [7], [9], [18]) 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 task observations.

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) 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.

VII. Related Work

Most studies about user involvement in practice explore the "early" phases of the software development lifecycle and specifically investigate how and to which degree users are involved [17], and which effects such involvement has on product acceptance [10]. 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. [9] found two main factors which constrain evolution decisions in development teams. 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 9 and 10), and additionally concludes that developers often lack the necessary information to be able to assess the stake size for a given feedback (Hypothesis

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 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 harden 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 [7] 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 1), and that there is no clear and common process on understanding users (Hypothesis 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 5 and 12). Moreover, the study revealed that determining the average end-user opinion is a hard task, which our interviews confirm (Hypothesis 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 14).

Zimmermann et al. [18] specifically focus on developers' problems with bug reports in open source projects. One of their main results 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 10). The authors further revealed a mismatch between information needed by developers and information which users actually provide, what intensifies our Hypothesis 4: According to our subjects, 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 reporting user (Hypothesis 17).

VIII. Conclusion

We reported on an empirical case study with five professional, small and medium-sized software development companies, exploring the current practice of user involvement during software evolution. We found that user feedback contains important information for developers, helps to improve software quality and to identify missing features. However, to assess its relevance and potential impact, developers need to analyze the gathered feedback, which is mostly done manually, in spite of its typically high volume – if it is done at all. Our results suggest that tools which consolidate, structure, analyze, and track user feedback would help to reduce the effort required by developers to deal with end user feedback. We claim that future work should aim at developing the corresponding tools to facilitate continuous user involvement. We suggest to perform empirical studies such as content analyses of user feedback to establish foundations and refine tool requirements.

Acknowledgement

Our thanks go to all study participants and to Walid Maalej, Helmut Naughton, Tobias Roehm, and the anonymous ICSE reviewers for their valuable feedback. This work was supported by the EC (FastFix project, grant FP7-258109).

References

[1] 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.

[2] 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.

[3] B. Flyvbjerg. Five Misunderstandings About Case-Study Research. *Qualitative Inquiry*, 12(2):219–245, Apr. 2006.

[4] 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.

[5] J. Grudin. Interactive systems: bridging the gaps between developers and users. *IEEE Computer*, 24(4):59–69, 1991.

[6] J. Grudin. Systematic Sources of Suboptimal Interface Design in Large Product Development Organizations. *Human-Computer Interaction*, 6(2):147–196, June 1991.

[7] 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.

[8] 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.

[9] 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.

[10] S. Kujala. User involvement: a review of the benefits and challenges. *Behaviour & information technology*, 22(1):1–16, 2003.

[11] 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.

[12] D. A. Norman and S. W. Draper. *User Centered System Design; New Perspectives on Human-Computer Interaction*. Erlbaum, 1986.

[13] 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.

[14] D. Pagano and W. Maalej. How Do Open Source Communities Blog? *International Journal on Empirical Software Engineering*, (May), 2012.

[15] C. Robson. *Real World Research*. Wiley & Sons, 3rd edition, 2011.

[16] 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.

[17] 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.

[18] 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.