2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing

On the Socialness of Software

Walid Maalej
Technische Universität München
Munich, Germany
maalejw@cs.tum.edu

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

***Abstract*—Conventional software engineering processes are rather transactional and lack a common theory for the involvement of users and their communities. Users are regarded as pure consumers, who are, at most, able to report issues. In the age of easy knowledge access and social media, discounting the users of software might threaten its success. Potentially valuable experiences and volunteered resources get lost. Frustrated users might even meet in social communities to argue against the software and harm its reputation.**

The goal of this research is to revolutionize the role of users, dissolving the boundaries to software engineers. We propose a novel framework for increasing the software socialness, being the degree of user and community involvement in the software lifecycle. Our framework consists of a benchmark, a process, and a reference architecture. The benchmark includes metrics for assessing and monitoring software socialness. The process enables engineering teams to systematically gather and exploit user feedback in the software lifecycle. The context aware reference architecture integrates social media into software systems and the engineering infrastructure. It observes users' interactions while they use the software and proactively collects in situ feedback.

***Index Terms*—Social media, user feedback, user community, product development, context awareness**

I. Introduction

In software engineering projects, single users are typically involved in eliciting and validating requirements. Communities of users are also increasingly receiving attention through active participation in open source projects, or through participation in user conferences and online support forums of popular software applications. However, there is no common and comprehensive theory for the involvement of users and their communities in the software lifecycle. Users are neither an integral part of the software engineering *processes*, nor of the software *systems* themselves.

On the one hand, software engineering processes are rather transactional and focus on a small number of representative users to give feedback in requirements engineering activities. Contributing to other activities such as testing, documentation, integration, or design is exceptional. On the other hand, user feedback mechanisms in software systems are not standardized and rather ad hoc – if they exist at all [7]. Typically the software "core features" are more important than those requested by users and described in communities. Communication channels that allow for collaboration among users or between users and developers are usually decoupled from a software system and its development infrastructure.

The lack of common means for a systematic user involvement in the software lifecycle has disadvantages:

- There are little indicators about the real software usage apart from download or sales numbers. Such indicators, if they exist, are marketing-oriented, expensive, and periodic. They focus on the opinion of single users and ignore how much influence users have among themselves and on the software. Consequently, instead of foreseeing trends software providers rather react on changing market characteristics and user requirements.
- In the age of Google and Wikipedia barriers for acquiring knowledge about the software and its technologies are low. Software vendors loose valuable resources, if knowledgeable users are unable to share their experiences. Driven by their own needs to improve the software, these users often develop macros, extensions, and workarounds that stay on their own machines.
- Exigent and frustrated users who feel their voices are ignored, are among the first who discontinue the usage of a software[1], especially if a simple web search leads to a "better" open source alternative. Such users can even harm the reputation of software. In the worst case, they organize Facebook and Twitter campaigns against the software instead of contributing to improving it[2].

This paper introduces a framework, which aims at revolutionizing the role of users in the age of social media and easy knowledge access. The goal of our framework is to increase the *socialness* of software, by making the involvement of users and user communities a first order concern of software systems and engineering processes. The framework consists of three parts, which represent the contributions of the paper. First, a benchmark and a set of measures allow vendors to monitor and assess the socialness of their software. Second, a social software engineering process model (called SNAIL) ensures the involvement of users in the software lifecycle. Third, a reference architecture enables the development and maintenance of social software involving users and communities.

Section II presents the benchmark and its measures. Section III describes the SNAIL process and gives examples how it can be applied. Section IV introduces the reference architecture and enabling technologies. Section V discusses related work and the feasibility of our framework. Finally, Section VI concludes the paper.

[1] for instance http://tech.fortune.cnn.com/2011/06/27/600-filmmakers-sign-complaint-about-final-cut-pro-x/

[2] for instance http://ihatelotusnotes.com or http://dreckstool.de/hitlist.do

978-0-7695-4612-4/11 $26.00 © 2011 IEEE
DOI 10.1109/DASC.2011.146

IEEE computer society

II. Benchmarking of Software Socialness

We define the *socialness* of a software system as the degree of involvement of its users and their communities in the software lifecycle. Software users can be involved either by actively working on a specific engineering activity, or by influencing a management or an engineering decision about the software. In the first case, users might suggest modifications and enhancements, perform tests, provide user support, maintain documentation, implement features, fix bugs, or organize events. In the second case, users might give feedback, vote for a specific decision, or influence the opinion of others. User communities represent an additional ground of the users' involvement on a group basis. Users externalize important knowledge in interest groups, and share their common interests about the software.

Socialness in practice ranges from little involvement (e.g. public bug tracker) to a complete involvement in open source communities. Figure 1 illustrates the categories 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 community as an indirect important means to create additional value. *Collaborative software* has few users, who are actively involved in the development process by providing ideas and partly conducting engineering activities. Finally, *social software* involves a large community of users, who actively contribute to the development of the software and pro-actively enlarge the community. Note that as opposed to collaborative software, *collaborative media* represent specific collaboration systems (e.g. Wikis or groupware). Similarly, as opposed to social software, *social media* represent specific systems for social networking and interaction (such as Facebook and blogs). Software can be collaborative or social, independently from its features and domain (e.g. social office, or social hotel booking system).

To assess socialness, we measure both the involvement of the single users and the community as a whole. We propose four measures for each dimension and semantic scales to quantify each of these measures (summarized in Table I).

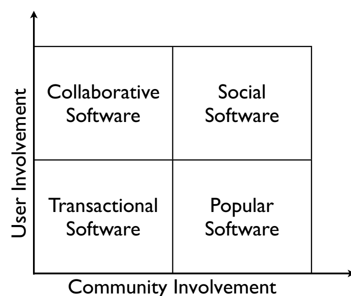

Figure 1. Benchmark for Assessing Software Socialness (following [5]).

A. Community Involvement Measures

Community involvement measures assess the sustainability of a user community's social structure, and how involved it is in the development decisions and activities.

a) Community Size: defines the number of known community members, including users (friends and followers) and contributors (fans and fanatics). The community is the basis for getting contributions and evidence of collected feedback. Collaborative web-based services such as Facebook, Amazon, or Google are typical systems with large communities[3].

b) Community Activity: denotes how active a community is in terms of communication volume and topic variation [2]. Communication volume is the total number of messages distributed by the community members during a specific time period. Topic variation is the number of topics and the distribution of messages among those topics each during a specific time period.

c) Community Interweaving: denotes the *ratio* of contributors in the whole community. Contributors actively participate in the software engineering process, either by performing particular activities such as development, management, and dissemination or influencing decisions about the evolution of the software. Typically, successful free and open source software has a high interweaving ratio as the developers are also users of the software. Gnome e.g. has more than 2,500 people committing source code to the repository [9]. Counting other users[4], who are writing or translating documentation, testing new features, reporting bugs, or answering emails, this number gets much larger.

d) Community Attractiveness: denotes the *growth* in terms of member gain and loss. It shows the ability of the software to attract and retain members to its community.

B. User Involvement Measures

User involvement denotes any kind of information exchange between the user and the engineering team. This includes friends and followers who just post comments on their blogs or accept to be observed to improve the software usage, as well as the active contribution via performing a particular activity.

e) Contribution Quality: distinguishes between *qualitative* and *quantitative* contributions. Qualitative contributions consist of subjective, "rich" data, such as words, pictures, and objects. Quantitative contributions lead to numbers and statistics. Typical qualitative contributions are source code artifacts, feature requests, or comments in product blogs. User polls and opinion surveys are typical quantitative contributions.

f) Contribution Explicitness: measures the willingness of users to contribute and is correlated to the contribution effort. User contribution might be *explicit,* if the user has the intent to provide the input, or *implicit*, if the user unintentionally provides information. Users can implicitly contribute, e.g. when they are observed in their environment to elicit useful information. Other examples are the collection of usage

[3]http://www.facebook.com/press/info.php?statistics
[4]http://www.gnome.org/get-involved/

Table I
SOCIALNESS OF SOFTWARE: COMMUNITY AND USER MEASURES.

	Measure	Question addressed	Value range
Community Involvement	Community Size	How many members does the software community have?	Few - Many
	Community Activity	How is the communication volume and topic variation in the community?	Low - High
	Community Interweaving	How is the ratio of contributors in the whole community?	Minority - Majority
	Community Attractiveness	How is the ratio of member gain and member loss?	Low - High
User Involvement	Contribution Quality	What is the quality of the contribution?	Qualitative - Quantitative
	Contribution Explicitness	Is the contribution intended?	Explicit - Implicit
	Contribution Time	Does the contribution occur during the user tasks?	Sync - Asynchronous
	Contribution Means	*Where* can users contribute?	Integrated - External

data, such as Eclipse Usage Data Collector, the Microsoft Customer Experience Improvement Program[5], and the Adobe Product Improvement Program[6]. Observing users during the usage of the software leads to a better understanding of their needs and the identification of new requirements. The scientific observation method is commonly used in behavioral science as well as in software engineering research.

g) Contribution Time: describes the synchronization of the user contributions. This ranges from *synchronous* interactions to *asynchronous* interactions. Asynchronous interactions result in retrospective contributions and user input. Issue trackers represent a typical example for asynchronous interactions. In contrast, with synchronous interactions the communication between users and developers occurs in real-time. For example, developers working on a particular bug might get their questions answered in real-time by the users. Consequently, users might get to know what the development team is currently working on. As more and more software gets connected and users become online, synchronous contribution is technically feasible. The advantage of synchronous contributions is that the communication happens in context. Users can report bugs or describe their needs, while actually using the software and not afterwards.

h) Contribution Means: describes whether users can provide feedback in situ or need to use a specific external tool or service. Contribution means are *integrated*, if users have the possibility to provide feedback while actually using the software, or *external*, if users have to leave the application and interrupt their workflows for contributing.

III. SNAIL: THE SOCIAL ENGINEERING PROCESS

Conventional development processes involve users only in a limited and transactional fashion. Although developers gather user input during various activities in requirements engineering, this feedback is collected separately from further development activities. Users and developers react on their respective feedback rather than collaborating directly. User communities, if they exist at all, are not promoted systematically.

We propose a social software engineering process called "SNAIL" that thoroughly and continuously involves users by establishing interaction channels and integrating user communities. Figure 2 depicts the main activities in the SNAIL process. Community activities as well as the interactions of the user with the application environment are continuously observed (snail tentacles). All other activities are performed in an iterative way (snail shell). Users may provide feedback about the system during its usage, and are proactively asked to do so in problem situations. Individual user feedback and community activities are analyzed systematically in order to filter, aggregate, and prioritize the information, and identify possible conflicts. The engineering team works on the feedback and uses community channels to connect to users for discussions and clarifications. Engineering and management decisions about the software are made on a social basis, considering the opinion of the community, and are communicated back to the community. Finally, updates and changes are published, restarting the cycle again. SNAIL can be regarded as an orthogonal extension to comprehensive development activities to increase their social nature and the socialness of developed software. In the following we discuss the main activities of SNAIL.

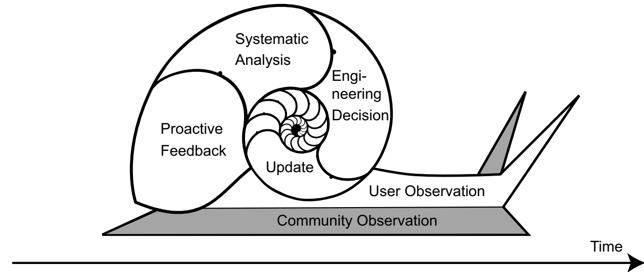

Figure 2. SNAIL - A Social Software Engineering Process.

A. User Observation

To improve the understanding of the circumstances under which users provide feedback, context information [7] (e.g. sequence of user interactions, artifacts used, or execution environment) is captured continuously and communicated to engineers (implicit contribution). This allows to support users when communicating issues, requesting changes requests, or rating features. User contributions thus emerge in situ (i.e. integrated means) while actually using the software, leading

[5]http://www.microsoft.com/products/ceip/EN-US/default.mspx
[6]http://www.adobe.com/misc/apipfaq.html

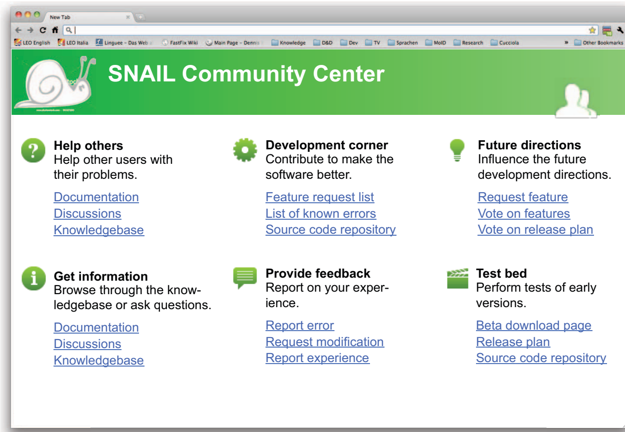

Figure 3. The SNAIL Community Center.

to a more intuitive and higher quality contribution. Finally, context information enables engineering team to reproduce and understand a given feedback.

Few systems already monitor their user interactions to facilitate their improvement and evolution. The Eclipse development environment for instance collects usage information[7] and sends it to the development team. This information is then used by developers, to e.g. optimize the software usability or to assess the importance of specific features, which can be helpful in prioritizing the development effort.

B. Community Observation

Users of social software have the possibility to organize themselves in communities, where they can share their experiences and discuss specific features. Social software integrates social media and proactively suggests to comment specific features or request other users' help in problem situations. Developers also participate in these communities to provide useful information, share insider knowledge, get insights, and directly communicate with users (community interweaving).

The user community should be systematically observed, due to the inherent dynamics of communities. Synergies between the opinions of different users emerge, leading to more perspectives, focused, and mature input. The "I like" button in Facebook or "+1" in Google Plus are examples in existing systems that foster these synergies while allowing for additional feedback. The resulting content is more focused, because users rather vote on existing content than providing new one. Additionally content becomes automatically ranked based on its acceptance in the community.

Figure 3 shows a mockup for the user community center of a fictitious system called the SNAIL environment. Users may request new features and modifications, vote on existing proposals, report errors, or externalize their experience with the system. These community actions are observed by the SNAIL environment to understand the community needs and expectations. Several existing systems provide online portals to enable similar community activities, mostly for support and marketing purposes. For example the Skype community portal[8] allows users to access the project knowledge base, suggest new features and modifications, or meet other users for general discussions.

C. Proactive Feedback

User feedback is collected continuously throughout the lifecycle of the social software. To this end, users are proactively asked to provide their feedback and input to improve the software. For instance, feedback can be explicitly requested if problem situations are detected while users interact with the software. Users may provide qualitative feedback (natural language), specify feedback category (e.g. modification or enhancement request), and select feedback type (e.g. positive or negative). Feedback data leverages the analysis of common usage patterns, helping to understand individual user behavior and problem situations.

There are few systems that proactively trigger actions from their users. Woogle [4] for example is a Wiki extension, that proactively asks users to contribute to specific topics where they are expected to have experience. These requests are based on the needs of other users for more information on these topics. With this technique Woogle reaches a more homogeneous match between needed and provided information.

D. Systematic Analysis

Gathered individual user feedback and community activities need to be analyzed in a systematic way to allow engineers to draw conclusions about the current usage status. Engineers for example need to know if there are common usage problems or errors that have to be solved. Managers on the other hand need to understand trends and sentiments or might want to assess the community involvement.

The goal of a systematic analysis is twofold. First, by filtering, aggregating, and prioritizing user feedback and community activities the amount of information for engineers will be reduced. Second, a systematic analysis will identify conflicts regarding the application under development. Conflicts arise since different users might have different, possibly conflicting preferences, needs, and expectations about the system.

Systematic analysis can deal with individual user feedback as well as community activities. Examples for analysis results include statistics about beta tests, documentation, ongoing discussions and sentiments, feature requests, error reports, experience reports, and social feedback on release plans. Google analytics[9] performs similar analyses on the user community of web pages. It includes statistics about the usage of a web page, grouped by characteristics of its users. Further, it provides conclusions about lost opportunities and success criteria based on when users leave web sites, which is an indicator for their sentiments.

[7] http://www.eclipse.org/epp/usagedata/

[8] http://community.skype.com/

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

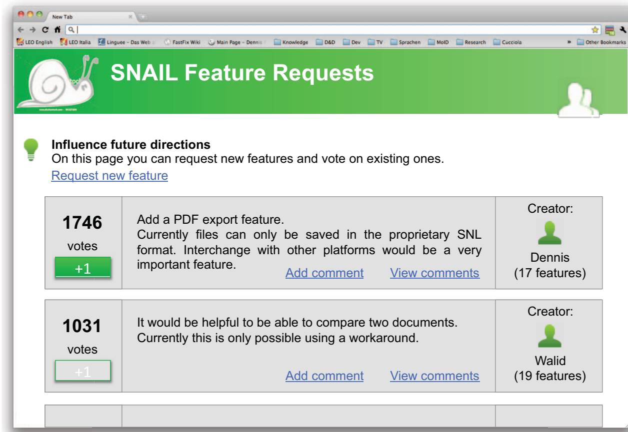

Figure 4. SNAIL Social Feature Requests.

E. Engineering Decision

The analysis and interpretation of individual user feedback and community activities triggers actions in several cases. For instance, if an error has been revealed it has to be fixed, or in the case of a high demand for a feature, the feature might have to be implemented, and release plans might have to be adapted. We subsume these actions under the term "engineering decisions". A social system should give users the possibility to influence engineering decisions by giving social feedback (i.e. vote, rate, or comment).

In Figure 4 we illustrate how feature requests can be prioritized by users, indicating the needs of the community, and thus influencing the decision which features to implement. The mockup is inspired by the existing commercial system UserVoice, which provides collective feature request and voting capabilities in the form of a social media enabled online community portal. UserVoice can be personalized for specific products, and reveals a prioritized list of the needs of the user community to the developers of the product.

F. Update

To create awareness about the impact of their contribution, users are informed about engineering decisions and rationale. The according information is sent back to the individual users as well as the community. Further, the changes to the software themselves are propagated back to the users – starting the SNAIL cycle again.

There are examples of similar activities in current systems. UserVoice for instance lets users know when the development of specific features they have proposed has started, and keeps them informed about the status. Moreover, several software vendors continuously give their users access to the latest software builds while in beta state.

IV. Reference Architecture for Social Software

In this section we describe a context-aware software framework that supports the social development process. The proposed framework enables integrated user contributions and fosters user communities for arbitrary software systems. Figure 5 shows an overview of the framework architecture and how it interacts with target application, engineering environment, and community media. The framework is designed using a client-server architectural style, with the *social engineering client* running together with the target application in the usage environment and communicating over the internet with the *social engineering center*. In the following we illustrate the main components that constitute the framework.

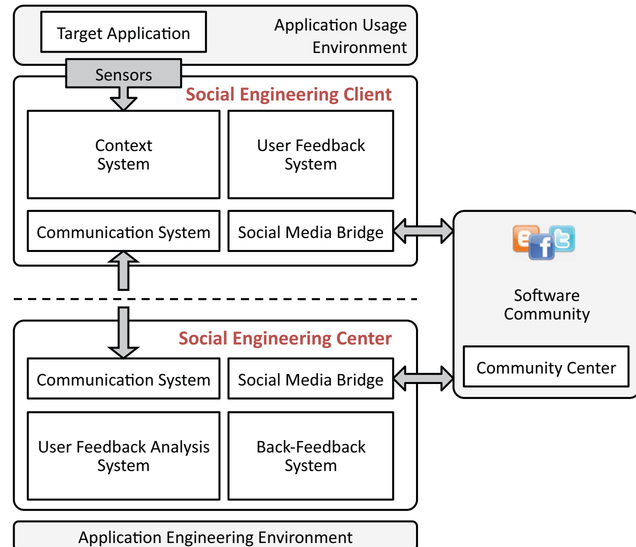

Figure 5. Software Framework Enabling Social Software Engineering.

A. Context System

The context system provides context elicitation and processing facilities. The elicited context includes information related to the user (e.g. user interactions) and the target application (e.g. application state). Hereby, the context system uses sensors to observe the target application, its runtime environment, and the operating system. Processing facilities then aggregate the collected context information and produce new semantically rich knowledge. For instance, user interactions are analyzed to identify repetitive patterns which could denote problem situations. Semantic web technologies (i.e. ontologies) are used to represent context information and facilitate its processing. Data mining and information retrieval methods, such as frequent pattern mining and clustering provide means to process the acquired context data.

B. User Feedback System

The user feedback system continuously and proactively collects individual feedback from users. User feedback can be triggered in three ways. First, by the context system, when it detects a problem situation based on the gathered context information (implicit contribution). Second, if the user explicitly wants to give feedback on a specific feature or situation. Finally, feedback can be requested on a regular basis,

Figure 6. SNAIL Feedback Report.

e.g. after a new release, after a modification, or after defined time periods. Users may provide feedback while actually using the application (integrated, synchronous contributions).

Figure 6 shows how an according feedback report for social software might look like. Users may provide qualitative feedback in the form of natural language, specify feedback category (e.g. modification or enhancement request), and select feedback type (e.g. positive or negative). Feedback is typically shared publicly in the community, but can also be selected to be private. Users can additionally select similar reports to facilitate feedback analysis and focus provided information. Moreover, users may tag the report to help other users find their feedback and further facilitate the analysis of the contribution. Tags are proposed based on already existing tags in the community. Finally, the user feedback system creates a report containing the specified feedback, including all specified information as well as relevant context information.

C. Community Center

A software community can emerge in principal in two different ways. Either with existing all-purpose social media systems such as online forums or Facebook groups (e.g. Microsoft Office[10]), or on a designated platform created and maintained by the engineers (e.g. Skype community). The community center represents the entrance point for the users to get involved in the software community in our reference framework. It provides facilities for users for instance to request new features or modifications, give social feedback on existing proposals, report errors, or externalize their experience with the system.

[10]http://www.facebook.com/Office

D. Feedback Channels

There are two channels for user feedback and back-feedback in the proposed framework. First, feedback can be communicated using the internal connection between the social engineering client and social engineering center. This channel allows the system to send gathered context information to the social engineering center and is in particular useful for private feedback. Second, feedback can be provided and discussed over social media channels (i.e. in user communities). This channel is suitable for public feedback and fosters important social properties (cf. Section II). Figure 6 illustrates how users may choose to publish their feedback in the user community.

E. User Feedback Analysis System

The user feedback analysis system systematically analyzes feedback provided by the users of the target application. This analysis can in principal be performed in four steps [8]. First, included domain concepts are identified using text mining and information retrieval techniques. In a second step, user feedback is aggregated according to the identified domain concepts, which can be done using unsupervised clustering techniques such as bisect k-means. To further reduce the amount of diverse information, user feedback can be filtered and prioritized in a third step. To this end, the feedback relevance for both users and engineers can be measured. In a fourth step, conflicting user preferences can be identified using social network analysis techniques. Group recommendation techniques can further support users proactively while creating their preferences to reduce or even avoid conflicts.

F. Back-Feedback System

The back-feedback system is in charge of providing feedback about the impact of their contribution back to the users. Further, it establishes a channel for clarification requests and discussions. To this end, it connects to the user community using the social media bridge, or communicates directly with the social engineering client of the specific user. Certain events may be of private interest for the user, and could be posted on the classic social media channels. For example "Walid received 10 kudos for proposing a feature that is now being released." or "Dennis just proposed a new feature for Open Office.".

G. Social Media Bridge

The social media bridge bundles all interfaces to communicate with different social media services in a standardized and extensible way. The social engineering client uses the social media bridge to publish user feedback in user communities and receives updates once a discussion emerges. In the social engineering center, the social media bridge also aids in the process of selecting responsible engineers for specific user feedback by connecting to the according network. Further, it establishes a connection to the user communities as well, to allow engineers to react on community feedback.

V. Discussion

A. Related Work

To our knowledge, there is no previous published research on making users and user communities an integral part of software applications and software engineering processes. However, there is relevant related work that either focus on in situ gathering of individual user feedback, or exploiting users' communities by using social media.

1) Gathering of User Feedback in Context: Recently several authors suggested to continuously and remotely gather user feedback, enabling software teams to adjust their applications to changing needs and requirements [7]. Seyff et al. [12] claim that future applications should support users in providing feedback about their needs *at runtime*. They introduce a mobile app called iRequire, which allows users to document their needs in situ, using photos, videos, sound, and text comments. The authors conducted a first study that shows the feasibility of tool-supported requirements elicitation. They envision self-adaptive systems that adapt to users' needs without the help of engineers. Similarly, Schneider et al. [11] propose a framework that allows users to provide feedback about large IT ecosystems as e.g. in a university. The authors suggest that users themselves annotate the type of feedback (e.g. complaint) and the context where the feedback applies (e.g. a particular subsystem) to facilitate analysis of the collected data. The framework also enables to collect physical context objects to help understanding and processing the feedback.

Our work is based on the same assumption that contextual information about the users is a major enabler for understanding their feedback. However, we also assume that the user's context and feedback must be correlated with the community dynamics for a qualitative and representative need. We suggest integrating the observation of both the community and the single user into the software engineering process by using social media and context frameworks. Moreover, a major difference of our approach is the integration of the feedback into the applications themselves, instead of using an additional tool for submitting needs enriched with contextual information. This makes the user feedback a part of the user's task and reduces the context switching to submit feedback. In addition to the physical context such as the location or a physical event (which can be captured in a movie), we suggest the work context as well as the user's characteristics such as capabilities and experience with the application. Finally, our approach also integrates the back communication channel from the engineering team to the users. Our goal is to make the users part of the software system as well as the software engineering process – i.e. tightening the socialness of software engineering.

2) Exploiting User Community: Related work on observing and analyzing users communities spans from studies which mine community artifacts in order to find hidden trends to systems that manages community contributions.

On the one hand, there exists a large body of research on extracting knowledge from community artifacts (such as forum discussions), including analyzing the semantics of the discussions, discovering trends, and identifying opinion leaders [13]. In a recent empirical study [9] we found that in their blogs open source developers mostly discuss requirements and product features with other developers and end users. Glance et al. [3] visualize the popularity of blog topics over time and show a correlation with real world trends. Pal et al. investigate the identification of themes and sentiments in social media by use of social network analysis and data mining [10]. We propose to exploit these results and mining techniques to communities of software applications and predict trends in application domains. These studies show that it is possible to extract users' opinions about specific products or features from user blogs, which supports the feasibly of our approach.

On the other hand, several social media enabled systems like UserVoice[11], Get-Satisfaction[12], and IdeaStorm[13] already allow users to collaboratively share new ideas and vote on existing suggestions. Users prioritize these requests according to the community needs. Bajic and Lyons [1] found that these collaborations focus users' efforts, which leads to more homogeneous feature requests and easier decision on how the application should evolve. With our approach users are spared explicitly entering feedback in a manual and transactional fashion. We suggest a process and an architecture for a proactive semi-automated sharing of feedback by collecting users' context and integrating it into the social media.

B. Feasibility and Challenges

We propose a social engineering process and a reference architecture that enable engineering teams to develop and maintain social software systems. To this end, we systematically combine activities that are already performed in existing engineering processes, yet in an isolated way. For instance, few systems observe individual users to allow engineers to optimize usability or prioritize feature requests (e.g. Eclipse usage data collector). Many software vendors observe and support online communities around their products to be able to assess sentiments and communicate directly with their users (e.g. Skype community portal). Moreover, users already may request and vote on new features, and thus influence engineering decisions (e.g. UserVoice). In open source communities such as GNOME, users even assume the role of engineers. Google Plus[14] or the Apple iOS SDK[15] represent examples for user involvement in software testing. The software is opened to the community while still in beta stage. Users then test the software while using it. We propose to systematically plan and conduct these activities and integrate them into software systems and engineering infrastructures. To evaluate the feasibility of this approach as a whole, researchers will have to carry out long term studies of according systems, users, and communities.

The general applicability of the proposed framework and the degree of software socialness that can be reached depend on (a) the system under development, (b) the type of users,

[11]http://uservoice.com, [12]http://getsatisfaction.com, [13]http://ideastorm.com
[14]http://plus.google.com, [15]http://developer.apple.com

and (c) on how these users can contribute. For instance, user involvement in security critical systems might not be suitable, whereas office tools and web applications can be improved by their user community.

The realization of our approach (in particular the implementation of the reference architecture) bears several technical and cultural challenges:

- Scalability. The quantity of user contributions in social systems imposes limits on how this information can be processed by engineers. With increasing number of users and frequent contributions manual analysis techniques become infeasible.
- Contribution Quality. If users contribute without professional support, resulting information content and quality is unpredictable, what can lead to misunderstandings [7], [14]. Users might express system properties using their own, possibly inhomogeneous terminology. In general, informal data like natural language is hard to analyze automatically because of its high degree of freedom.
- Conflicting Contributions. Different users will have different, possibly *conflicting* preferences and expectations. To make decisions from this data, conflicts have to be identified and resolved. Continuous user contributions in social systems can lead to frequent conflicts, complicating manual identification and resolution techniques.
- Integration. Involving users and their communities requires a high integration effort. Several systems and technologies, including users' environments, social media, and engineering environments have to be monitored and integrated. These systems might run on different platforms, have different interfaces, and contradicting licenses.
- Privacy, Usability, and Incentives. Collecting context information raises questions about privacy and control. The challenge is therefore to use sensitive information and ensure that it will not be abused. By doing so, usability trade-offs might emerge (e.g. the user is asked to confirm particular operations) possibly leading to less incentives for contributing useful information.
- Company Culture. Involving users and communities in engineering activities represents a paradigm shift for most conventional and in particular commercial organizations. The acceptance of the presented model depends on the company culture and is correlated with emerging benefits.

The main building blocks have already shown to be technically feasible. In TeamWeaver [6] we implemented a context elicitation and interpretation framework for the domain of software development itself. We instrumented the work environment of developers, such as code editors, email programs, browsers, and test tools. The collected data is used to trigger recommendations (proactive feedback) for sharing and accessing useful information. Moreover, existing research successfully addresses challenges of analyzing user contributions in large-scale requirements engineering settings [8].

VI. Conclusion

We aim at increasing the socialness of software, making user and community involvement a first order concern in software lifecycles. To this end we propose a benchmark, a process model, and a reference architecture. The benchmark includes metrics for assessing the socialness of a software system. The engineering process SNAIL systematically observes single users and their communities, enabling developers to systematically gather and exploit contributions in the software lifecycle. The reference architecture enables the development and maintenance of social software. Future challenges for researchers and practitioners include conflicting contributions, privacy, and the acceptance in conventional organizations.

Acknowledgement

We thank Raian Ali and Hans-Jörg Happel for their valuable feedback. This work has been supported by the FastFix project, which is funded by the EC, grant agreement no. FP7-258109.

References

[1] 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. ACM, 2011.

[2] B. S. Butler. Membership Size, Communication Activity, and Sustainability: A Resource-Based Model of Online Social Structures. *Information Systems Research*, 12(4):346–362, Dec. 2001.

[3] N. Glance, M. Hurst, and T. Tomokiyo. BlogPulse: Automated trend discovery for weblogs. In *WWW 2004 Workshop on the Weblogging Ecosystem: Aggregation, Analysis and Dynamics*, volume 2004, 2004.

[4] H.-J. Happel. Social search and need-driven knowledge sharing in wikis with woogle. In *Proceedings of the 5th International Symposium on Wikis and Open Collaboration*, WikiSym '09, pages 13:1–13:10, New York, NY, USA, 2009. ACM.

[5] B. Libert. *Social Nation: How to Harness the Power of Social Media to Attract Customers, Motivate Employees, and Grow Your Business*. Wiley, 2010.

[6] W. Maalej and H.-J. Happel. A lightweight approach for knowledge sharing in distributed software teams. In *7th International Conference on Practical Aspects of Knowledge Management*, volume 5345 of *Lecture Notes in Computer Science*, pages 14–25. Springer Verlag, 2008.

[7] W. Maalej, H.-J. 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, New York, NY, USA, 2009. ACM.

[8] D. Pagano. Towards Systematic Analysis of Continuous User Input. In *Proceedings of the 4th International Workshop on Social Software Engineering*. ACM, 2011.

[9] D. Pagano and W. Maalej. How Do Developers Blog? An Exploratory Study. In *Proceedings of the 8th Conference on Mining Software Repositories*. ACM, 2011.

[10] J. Pal and A. Saha. Identifying Themes in Social Media and Detecting Sentiments. In *2010 International Conference on Advances in Social Networks Analysis and Mining*, pages 452–457. IEEE, Aug. 2010.

[11] K. Schneider, S. Meyer, M. Peters, F. Schliephacke, J. Mörschbach, and L. Aguirre. *Feedback in Context : Supporting the Evolution of IT-Ecosystems*. Springer Berlin / Heidelberg, 2010.

[12] N. Seyff, F. Graf, and N. Maiden. Using Mobile RE Tools to Give End-Users Their Own Voice. In *Requirements Engineering, IEEE International Conference on*, pages 37–46, 2010.

[13] X. Song, Y. Chi, K. Hino, and B. Tseng. Identifying opinion leaders in the blogosphere. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 971–974, New York, New York, USA, 2007. ACM.

[14] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss. What Makes a Good Bug Report? *IEEE Transactions on Software Engineering*, 36(5):618–643, Sept. 2010.