Influencing the Adoption of Software Engineering Methods Using Social Software

Leif Singer, Kurt Schneider
Software Engineering Group
Leibniz Universität Hannover
Hannover, Germany
{leif.singer, kurt.schneider}@inf.uni-hannover.de

***Abstract*—Software engineering research and practice provide a wealth of methods that improve the quality of software and lower the costs of producing it. Even though processes mandate their use, methods are not employed consequently. Software developers and development organizations thus cannot fully benefit from these methods. We propose a method that, for a given software engineering method, provides instructions on how to improve its adoption using social software. This employs the intrinsic motivation of software developers rather than prescribing behavior. As a result, we believe that software engineering methods will be applied better and more frequently.**

Social Software; Motivation; Adoption; Process; Virtual Communities; CSCW; Social Network Sites

I. Introduction

Many software engineering methods, such as unit testing or version control, have been shown to positively influence the quality and costs of the produced software. Practitioners and researchers alike are constantly refining existing methods and inventing new ones.

However, in practice, many of these are often not used optimally or not used at all. Anecdotally, we have seen this happen in industry projects. Test cases don't get written because of tight deadlines. Code exhibits high coupling, even though developers know better. Employees perform version control by emailing compressed project directories back and forth. Using processes helps tackling these problems to a certain degree – but mandatory procedures are not always followed and may be met with resistance [11]. Key motivations for software engineers are autonomy, independence, and being included in decision making [3] – a stark contrast to activities mandated by processes.

In the past years, much research on the effects of social software has been published. Some of these results stem from computer science, but many have their roots in social psychology and group dynamics. They show that social software helps motivating users and can propagate behavior and information. For example, users of social network sites are more likely to enact a certain behavior if they were able to observe their contacts exhibiting the behavior before [5, 6]. Social software may also help raising awareness in project teams, as it makes visible what is happening in the project [16].

Software engineering – a social activity – could benefit from the systematic application of social software. In particular, we believe that it can influence the use of software engineering methods. This is currently not yet done systematically; rather some tools already use these effects in some places. We present the first results of a systematization of this approach and describe our plans for future work in this area.

II. Related Work

We are creating a systematic method that can be applied to a multitude of software engineering methods, i.e., it is method-independent. Its goal is improving the use of these methods. Software development processes, such as the V-Model XT, as well as software process improvement models, have similar goals and properties.

The latter were first meant to assess to what degree an organization implemented a given development process. By now, they are also used to improve the implementation of these processes in organizations. Examples are CMMI [15] and SPICE [12].

However, software engineers do not accept processes just because they are mandated. Riemenschneider et al. found out that „individual developer acceptance is far from assured even in the presence of an organizational mandate“ [11]. Beecham et al. later found out that one of the key motivators for software engineers seems to be autonomy. They have a wish for independence, yet want to be included in decision making [3]. Simply mandating a process doesn't achieve any of these things.

Contrary to processes and process improvement models, our approach strives to support the existing motivations of software engineers. In our view, this would complement processes and developer education to improve the adoption of software engineering practices.

An example that plainly shows how existing software engineering methods can be improved upon using social software features is Brun et al.'s *Crystal* [4]. When developers using a version control repository work on the same branch of a project simultaneously, conflicts can easily arise. Crystal proactively monitors the repository and warns collaborators when it detects potential for future conflicts.

The problem identified with the software engineering method of version control was that developers would notice conflicts rather late, so that they were harder to fix than if they had been addressed earlier. To solve this, Crystal has an internal model of the *group* of the *users* involved with a project, i.e., the software engineers. From commit *events*, it

derives a conflict potential for each user, which we call *system-derived information*. As all developers are *subscribed* to changes in that status, Crystal sends them *notifications* about these changes – and, therefore, about potential conflicts. As we will see in the following section, these are all elements of social software.

When given the software engineering method *version control* and the objective of *resolving conflicts earlier*, our method would provide its user with a set of guidelines regarding potentially helpful effects from social software, and a strategy as to how to combine and integrate them with version control. The end result should be a solution similar to Crystal.

To make this possible, the following section first systematizes elements of social software and then provides an overview of the results of our preliminary literature review regarding the effects of such elements.

III. A Systematization of Social Software

Using a set of criteria, we analyzed a set of software applications for *social application dialogs*. Using qualitative coding, we extracted patterns and categories from the raw data. As this procedure is not the main focus of this paper, we now only provide a sample of what we found for illustrative purposes.

A. The Elements of Social Software

The user represents the individual using the software. To other users, they're often represented in a *user profile*. A *situational profile* might contain additional information that is only valid for a certain timeframe, such as the user's location or other individuals they're currently with. An *embedded profile* is used to represent the user in contexts where the user isn't the main subject. For example, the user's name and photo, linking to her profile can be used when attributing content. *Mentions* of users appear either interwoven into or as metadata added to other content. These are often just the user's name with a profile link.

Users can form **relationships** with each other, which can either be one-sided or mutual and may or may not need to be accepted by the other party. Facilities that help users *find people* enable them to create new relationships. For organizing one's relationships, systems may provide *groups*, whose existence may be private or public. Having a relationship to a user often means *subscribing* to the content provided by that user.

The content created by subscribed users is often shown in a chronologically ordered digest, a *stream*. Content may either be created by a user explicitly, or be derived by the system from the user's *activities* or other *events*. Depending on the goals of the system, content is created in a *fire & forget* manner or is *maintained* for a longer period of time. A special case is content that is used for *collaboration*. While either content can be directed at another user via *mentions*, *messages* are private content exchanged between two users. The system notifies its users of interesting content using *notifications* – while each system may define what is interesting on its own, notifications are often triggered by other users' interaction with the user via content or relationships. Examples are new friend requests, a private message, a mention, or a comment on content the user posted before. Among other forms of metadata, *system-derived information* may be shown together with content or users, such as the number of comments addressing the content. Often, this is used as a *content hint* that leads the user to more detailed views of the information.

Content can often be **annotated**. Apart from *comments* that allow users to discuss content, mechanisms for *rating* content may exist. Depending on the systems, these can range from a very low-barrier implementation ("like") to those requiring directed effort on the rater's part (e.g., a book review).

To make the propagation of content through the network of users easier, content can be **shared**. This can either be *directed sharing*, targeting one or more specific users, or may be a simple *repost* of existing content, distributed to all the user's subscribers. For some implementations, the user is allowed to add an own comment on the content to the sharing act. Public, undirected *reposts* can also be interpreted as a kind of rating, stating, *"I approve of this content."*

Social software systems emit **events**. When a given *condition* is satisfied, an event may trigger an *action* by the system. That action might generate *content* on behalf of a user ("Jane changed her profile picture"), send a *notification* to one or more users ("John subscribed to your posts"), or generate a change in a *system-derived status* ("Jack just earned a badge").

The signals that get sent from a social software system to its users are thus mostly *activities*, *notifications*, and *system-derived information*.

B. The Effects of Social Software

In a preliminary literature review, we found several effects that can be supported or triggered by social software. Here, we provide a sample to illustrate our findings.

1) Information Spread

Social network sites often have a mechanism that allows users to *reshare* content they obtained from other users. On Facebook, this is achieved using the *Like* and the *Share* functions. Twitter has a *Retweet*, while Google+ also calls it a *Share* or *Reshare*. These allow content to jump from one social network to another, enabling content to "go viral".

For Twitter, Kwak et al. found that Retweets allow users to spread information very far, largely independent of a user's follower count [10]. For Facebook, Sun et al. found that large "viral" chain reactions of information diffusion do not start with a single user's post [14]. Instead, it takes several users posting the same content independently to form large diffusion clusters.

2) Behavior change through activity awareness

Most social network sites contain a stream that shows the posts and activities of a user's contacts. Being exposed to the activities of peers and being able to discuss them seems to support the spread of behavior among users.

For example, Foster et al. designed a Facebook application that allowed study participants to enter their daily step counts, taken with a step counting device throughout the day [8]. For some users, the application would automatically

create a post stating the step count for that day. This led to a significant increase in step activity compared to the participants whose application did not create those posts.

Centola shows that being exposed to the activities of one's contacts increases the likelihood of adopting the observed behavior – even if these contacts were randomly selected [6].

Using data from Facebook, Burke et al. found the same effect: new users of the site were more likely to share content themselves when they saw their contacts do so [5].

3) Gamification for increasing motivation

According to Deterding et al., *gamification* is "the use of game design elements in non-game contexts" [7]. A *game design element* might be anything in the spectrum from a leaderboard that ranks the system's users, to the use of actual game design methods such as carefully crafted storytelling. Several effects have already been successfully shown to work.

For example, Antin and Churchill discuss the use of *badges* in social media and derive their functions [1]. Among the five functions they find, they identify three that have a social component to them. *Reputation* allows others to classify a user based on what the badges represent, which might be, e.g., skill, interests, or experience. Badges also serve a *status* function, in that those who earned them might see them as status symbols with regards to others. Finally, badges can increase *group identification*, which has a positive effect on cooperation.

Even from our unstructured first literature review, we found many more documented effects of social software. We plan to conduct a systematic literature review on effects such as the above to achieve a more complete picture. Our method, as presented in the next section, will systematize the usage of these effects to augment software engineering methods.

IV. Augmenting Software Engineering Methods

The inputs to our method are a *software engineering method* (*SE method*) and a *goal* that should be reached with regard to that method. We do not include finding that goal – established methods such as GQM [2] can be used. This seems especially beneficial, as the execution of the method may require finding metrics that describe the goal.

Also, a set of *users* is required that use or should use the SE method. These may have certain characteristics that our method will need to consider. For example, different effects might be suitable for users of different education backgrounds. Between these users, *relationships* needs to be derivable – e.g., the fact that they work for the same organization or are involved with the same project.

There are several options for recreating these relationships in software. They may be created automatically when setting up the system. In this case, the users could or could not be allowed to change them. Alternatively, one could let the users create their relationships all by themselves. The latter option is closer to the organic growth typically associated with social software. Depending on the situation, the options need to be balanced against each other. Our method will contain decision helps for this, which we have yet to extract from literature.

The output of the method is a set of instructions that describe how the SE method should be augmented with social software and which effects should be achievable by that. Figure 1 illustrates the course of action for our method.

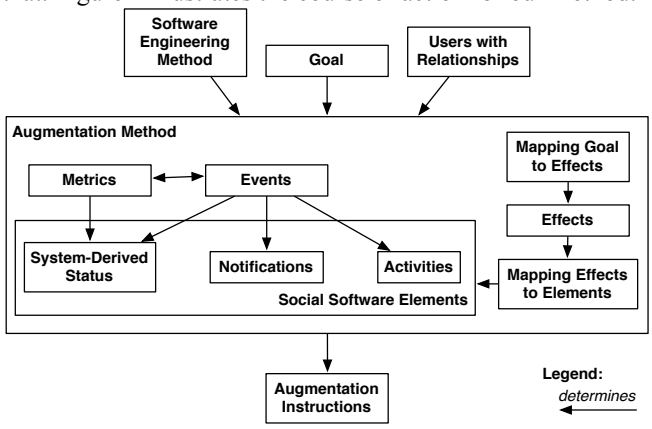

Figure 1. Overview of the proposed augmentation method.

A. Mapping a goal to the effects of social software

The goal that is to be achieved with regard to the SE method may take one of several forms. A goal could be to

- introduce a new behavior;
- increase or decrease the frequency of a behavior;
- increase or decrease the number of users exhibiting the behavior;
- stop a behavior;
- increase or decrease the quality of a behavior.

This goal needs to be transformed into a set of social software effects that would help in achieving the goal. We already found several such effects in literature. Our method will include a mapping that provides a set of desirable and achievable effects for a classified goal. As our method is a work in progress, this mapping is still unfinished.

However, an example situation would be to increase the number of users exhibiting a behavior. The mapping would then provide an effect that transmits existing behavior from a set of users to a different set of users not yet exhibiting the behavior. Burke, Marlow and Lento showed that this is indeed an effect that can be achieved with social software [5].

B. Mapping effects to social software elements

Once the desirable effects have been found out, another mapping should provide sets of social software elements that may help in creating the effects. We will derive this mapping from literature as well. For the example above, the elements needed to achieve the effect would be a *stream* where users are made aware of each other's actions and *activities*, i.e., content displayed in the stream as if it was created by the user exhibiting the behavior, but ultimately automatically derived by the system.

C. Metrics & events

To express the goal in the system and to implement it using social software, one or more *metrics* might be needed. These should be able to describe quantitatively what should be achieved. Also, it might be necessary to generate *events* whenever the desired behavior is performed.

If, for example, more commits should be submitted to the version control system, the commits themselves could be the events. A suitable metric would be the number of commits for a time span. In the context of Crystal and conflict prevention, the metric would measure the probability of a conflict for a user at each point in time. Every time a user commits to the repository, the metric for the user would be updated. The commit would then be an event that influences the metric. In this case, however, the event doesn't address the desired behavior yet – resolving conflicts more frequently. Therefore, we would derive another class of events from the metric: each time a certain probability threshold is reached, an event would be emitted. Metrics and events influence each other.

D. Notifications, Activities, & System-Derived Information

As we have argued in section III, *notifications*, *activities*, and *system-derived information* are those social software elements that enable the system to send signals to the user. We derive them from metrics and events.

Notifications inform the user of important events in the system, i.e., those in which the user has an interest. These may be things the user regards as positive – such as new positive ratings for her content – or as requiring action – such as due dates or problems that can still be dealt with.

Activities are a kind of content that is generated by the system in place of the user. For a commit to version control, this could a short message that contains the commit message and affected files. This makes accessing this data easier for the user's contacts.

System-derived information is often presented in relation to a user or content. Examples are rating content, the number of contacts for a user, or a rank that a user might have reached. Many variations are possible, each creating different effects.

For these social software elements, our method will provide realization recommendations and, based on literature, mention advantages and disadvantages of these solutions with regard to the intended effects.

V. Conclusions & Outlook

We presented a draft of our method. We believe it will allow software development organizations to systematically influence the adoption of software engineering methods by software engineers. It is based on several published effects that can be achieved by social software.

Our next step regarding the theoretical basis of our method is to derive the list of effects more systematically. For this, we will conduct either a systematic literature review as proposed by Kitchenham [9].

To evaluate our method, we conducted an experiment with computer science students, in which we aimed to motivate more frequent and smaller commits to our version control system [13]. We used our method to find out which kinds of helpful effects social software might be able to provide. We then provided the students with a web-based stream of commit activities and additional derived events. We used notifications to remind them of their teams activities and presented them their team's total number of commits. While our results look promising so far, we will conduct more experiments to iteratively refine our method.

References

[1] J. Antin and E. Churchill. Badges in Social Media: A Social Psychological Perspective. In *CHI 2011 Gamification Workshop Proceedings*, Vancouver, BC, Canada, 2011.

[2] Victor R. Basili. Software modeling and measurement: the Goal/Question/Metric paradigm. Technical report, College Park, MD, USA, 1992.

[3] S. Beecham, N. Baddoo, T. Hall, H. Robinson, and H. Sharp. Motivation in Software Engineering: A systematic literature review. *Information and Software Technology*, 50(9-10):860–878, 2008.

[4] Yuriy Brun, Reid Holmes, Michael D. Ernst, and David Notkin. Proactive detection of collaboration conflicts. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, ESEC/FSE '11, pages 168–178, New York, NY, USA, 2011. ACM.

[5] M. Burke, C. Marlow, and T. Lento. Feed me: motivating newcomer contribution in social network sites. In *Proceedings of the 27th international conference on Human factors in computing systems*, pages 945–954. ACM, 2009.

[6] D. Centola. The spread of behavior in an online social network experiment. *Science*, 329(5996):1194, 2010.

[7] Sebastian Deterding, Dan Dixon, Rilla Khaled, and Lennart Nacke. From Game Design Elements to Gamefulness: Defining "Gamification". In *Proceedings of MindTrek'11*. ACM, 2011.

[8] D. Foster, C. Linehan, B. Kirman, S. Lawson, and G. James. Motivating physical activity at work: using persuasive social media for competitive step counting. In *Proceedings of the 14th International Academic MindTrek Conference: Envisioning Future Media Environments*, pages 111–116. ACM, 2010.

[9] Barbara Kitchenham. Procedures for Performing Systematic Reviews. Technical Report Keele University Technical Report TR/SE-0401, Software Engineering Group, Department of Computer Science, Keele University, 2004.

[10] H. Kwak, C. Lee, H. Park, and S. Moon. What is Twitter, a social network or a news media? In *Proceedings of the 19th International World Wide Web Conference*, pages 591–600. ACM, 2010.

[11] C.K. Riemenschneider, B.C. Hardgrave, and F.D. Davis. Explaining software developer acceptance of methodologies: A comparison of five theoretical models. *IEEE Transactions on Software Engineering*, 28(12):1135–1145, 2002.

[12] Automotive SIG. *Automotive SPICE® Process Reference Model*. The SPICE User Group, 2010.

[13] Leif Singer and Kurt Schneider. It was a Bit of a Race: Gamification of Version Control. In *Proceedings of the 2nd international workshop on Games and software engineering (in press)*, 2012.

[14] E. Sun, I. Rosenn, C. Marlow, and T. Lento. Gesundheit! modeling contagion through facebook news feed. In *Proceedings of the International AAAI Conference on Weblogs and Social Media*, 2009.

[15] SCAMPI Upgrade Team. Appraisal Requirements for CMMI, version 1.1. Technical Report CMU/SEI-2001-TR-034, Carnegie Mellon University Software Engineering Institute, 2001.

[16] C. Treude and M.A. Storey. Awareness 2.0: Staying aware of projects, developers and tasks using dashboards and feeds. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, 365–374. ACM, 2010.