7th International Workshop on Social Software Engineering (SSE 2015)

Proceedings

Imed Hammouda and Alberto Sillitti

September 1, 2015
Bergamo, Italy

The Association for Computing Machinery, Inc.
2 Penn Plaza, Suite 701
New York, NY 10121-0701

Notice to Past Authors of ACM-Published Articles

ACM intends to create a complete electronic archive of all articles and/or other material previously published by ACM. If you have written a work that was previously published by ACM in any journal or conference proceedings prior to 1978, or any SIG Newsletter at any time, and you do NOT want this work to appear in the ACM Digital Library, please inform permissions@acm.org, stating the title of the work, the author(s), and where and when published.

ACM ISBN: 978-1-4503-3818-9

Additional copies may be ordered prepaid from:

	Phone: 1-800-342-6626
ACM Order Department	(U.S.A. and Canada)
P.O. BOX 11405	+1-212-626-0500
Church Street Station	(All other countries)
New York, NY 10286-1405	Fax: +1-212-944-1318
	E-mail: acmhelp@acm.org

Production: Conference Publishing Consulting
D-94034 Passau, Germany, info@conference-publishing.com

Message from the Chairs

The Workshop on Social Software Engineering (SSE) focuses on the interplay between social computing and software engineering. On one hand, social factors in software engineering activities, processes and tools are useful for improving the quality of development processes and the software produced by them. Examples include the role of group awareness and multi-cultural factors in collaborative software development. On the other hand, social software mediates people-to-people communication, supporting human choices, actions and interactions with each other. Social software needs to accommodate a wide range of social concepts, such as trust, governance, reputation, and privacy. Being social, the software would also need to be receptive to users' choices and give them a voice in the design, operation and evolution decisions. The SSE workshop brings together academic and industrial perspectives to provide models, methods, tools and approaches to address these issues.

SSE has been held in Munich, Kaiserslautern, and Paderborn in conjunction with the Software Engineering conference (SE'08, SE'09, and SE'10) and in Szeged in conjunction with ESEC/FSE 2011. In 2009, we merged the original community with the SENSE workshop (Software ENgineering within Social Software Environments). In 2010, we merged with the SoSEA workshop (Social Software Engineering and Applications) held at ASE'08 and ASE'09, and called the conjoined workshop Social Software Engineering. At ESEC/FSE 2013, we combined with Web2SE workshop (the International Workshop on Web 2.0 for Software Engineering). Web2SE was a very successful workshop, hosted at ICSE (the International Conference on Software Engineering). Web2SE brought together researchers studying how social software impacted software engineering, and promoted ways in which Web 2.0 approaches and technologies could open up new opportunities for developers to form teams and collaborate. SSE'14 was collocated with ESEC/FSE in Hong Kong.

SSE'15 distinguishes between two kinds of papers: Full papers (which describe authors' novel and validated research work), Short position papers (which describe a new idea or work in progress). This year, we received 9 submissions from 4 countries (Finland, Germany, Italy, and Sweden). After three members of the Program Committee reviewed each submission, we selected 8 papers (5 full papers and 3 as short position papers). Papers were selected based on originality, quality, soundness and relevance to the workshop.

The fundamental objective of SSE is to socialize the software engineering process and to find novel ways for engineering the social features of software. We advocate openness to the individual and collective contribution of people and the crowd to support software design decisions. We also call for methods and mechanisms for software systems which are able to accommodate the wide range of social requirements of their stakeholders and users.

Imed Hammouda, Chalmers and University of Gothenburg, Sweden
Alberto Sillitti, Center for Applied Software Engineering, Free University of Bolzano, Italy

SSE 2015 Organization

Organizing Committee

- Imed Hammouda, Chalmers and University of Gothenburg, Sweden
- Alberto Sillitti, Center for Applied Software Engineering, Free University of Bolzano, Italy

Steering Committee

- Raian Ali, Bournemouth University, UK
- Andrew Begel, Microsoft Research, USA
- Imed Hammouda, Chalmers and University of Gothenburg, Sweden
- Filippo Lanubile, University of Bari, Italy
- Walid Maalej (chair), University of Hamburg, Germany

Program Committee

- Raian Ali, Bournemouth University, UK
- Rami Bahsoon, University of Birmingham, UK
- Andrew Begel, Microsoft Research, USA
- Kelly Blincoe, Auckland University of Technology, New Zealand
- Mohamed Amine Chatti, RWTH Aachen University, Germany
- Fabio Calefato, University of Bari, Italy
- Fabiano Dalpiaz, Utrecht University, The Netherlands
- Rosalba Giuffrida, IT University of Copenhagen, Denmark
- Imed Hammouda, Chalmers and University of Gothenburg, Sweden
- Timo Johann, University of Hamburg, Germany
- Gail E. Kaiser, Columbia University, USA
- Eric Knauss, University of Gothenburg, Sweden
- Filippo Lanubile, University of Bari, Italy
- Kelly Lyons, University of Toronto, Canada
- Dennis Pagano, CQSE GmbH, Germany
- Keith Phalp, Bournemouth University, UK
- Rafael Prikladnicki, PUCRS, Brazil
- Norbert Seyff, University of Zurich, Switzerland
- Alberto Sillitti, Alberto Sillitti, Center for Applied Software Engineering, Free University of Bolzano, Italy
- Leif Singer, iDoneThis.com, USA / University of Victoria, Canada
- Margaret-Anne Storey, University of Victoria, Canada
- Damian Andrew Tamburri, Vrije University Amsterdam, The Netherlands
- Christoph Treude, McGill University, Canada

Contents

Communicating Software Testing Culture through Visualizing Testing Activity

Raphael Pham
Software Engineering Group
Leibniz Universität Hannover
Hanover, Germany
Raphael.Pham@inf.uni-hannover.de

Jonas Mörschbach
Leibniz Universität Hannover
Hanover, Germany
Jonas.Moerschbach@se.uni-hannover.de

Kurt Schneider
Software Engineering Group
Leibniz Universität Hannover
Hanover, Germany
Kurt.Schneider@inf.uni-hannover.de

ABSTRACT

Inexperienced developers, for example new graduates joining a software development company, have difficulties applying their software testing knowledge. They lack hands-on experience and often have a dismissive attitude towards systematic testing, which hinders their adoption of testing activities. If the novice cannot quickly adopt a healthy testing culture during the onboarding phase, her progress as a high-quality engineer may be hindered. Here, cues from social coding sites can potentially help: Simple signs of a team's testing culture can facilitate more testing by contributors. We propose to make the team's testing culture visible by strategically employing traits of social transparency. We introduce six dashboard-like testing signals into the novice's IDE and prominently display how senior developers are testing. A preliminary evaluation with 24 soon-to-be Bachelor graduates showed encouraging results: Being reminded of their lagging test progress induced a need to test more. Visual comparison to colleagues' testing performance woke competitive feelings in students and made them want to write more test.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*Testing tools*; D.2.9 [**Software Engineering**]: Management—*Programming teams*

General Terms

Management, Human Factors

Keywords

Testing Culture, Newcomers, Social Transparency

SSE'15, September 1, 2015, Bergamo, Italy
© 2015 ACM. 978-1-4503-3818-9/15/09...$15.00
http://dx.doi.org/10.1145/2804381.2804382

1. INTRODUCTION

Quality assurance measures often boil down to some form of systematic testing — often automated testing. Thusly, software testing abilities are in high demand in the current software engineering industry. However, in a previous study [15], we have found that soon-to-be Bachelor graduates have considerable problems with systematically testing and often approach it with a dismissive attitude. About 100 students had just finished a four month software project in which they were asked to prove their software engineering skills — including testing skills. When they added a test suite very late in the project, it did not find any faults — most faults had been found in laborious manual testing during the course of the project. This left them with a negative impression on automation of testing. This is problematic: new graduates form the basis of our software engineering workforce. After finishing their studies, they will apply for software engineering positions at software development companies around the globe. [1]

The onboarding phase is quite turbulent for young new hires. They have to learn how the company develops software while adopting new work-related values. A lack of testing skills and dismissive attitude towards testing can be hindering in this situation and eventually keep the applicant from becoming a high-quality developer. Depending on the size of the company and the resources available, practitioners teach new hires how to code and test (— while testing often has to be taught from scratch). Different measures such as coding boot camps or mentoring efforts, however, are costly and time consuming. Not all companies are willing to invest in new hires this way. A need to support both the inexperienced developer and the practitioner during this onboarding phase arises. Our work focusses on how to sensitize inexperienced developers — such as graduates — for the testing activities in their team. This way, we target to improve on their dismissive attitude towards automated testing and elevate its perceived importance. **Eventually, we aim to increase testing activity in inexperienced new hires while keeping the instructional efforts on the practitioner's side low.** Here, social transparency mechanisms can potentially help: In a previous study [16], we observed that a high degree of social transparency on the social coding site GitHub.com positively influenced the test-

[1] In an ongoing study, we are investigating whether practitioners have recognized a significant lack of testing skills in applicants from university. Preliminary results indicate dissatisfaction among practitioners of our study's population.

ing behavior of code contributors. [2] When users saw that another user's project used a popular CI-service, indicated by a badge on the project's profile site, they felt obliged to add written tests to their own contributions. The same effect took place, when users saw a running test infrastructure and a test suite in other users' projects. Additionally, contributors used other users' written tests as a basis for their own tests — copying and pasting helped them in writing their own tests. We dubbed such visible cues of testing activity in a project *testing signals*.

This work is a first step towards employing social transparency effects in order to increase testing activity in inexperienced developers. Our goal is to make onboarding more efficient by getting the newcomer to write more tests early on. We want to archive this goal by increasing the visibility of testing signals and making the newcomer aware of the testing activities around her — leading to a faster adoption of systematic testing.

We present a first reference model for a so-called *testing signal* (Section 5) together with six testing signals (Section 6) and a multi-staged preliminary evaluation of their effects (Section 7) on inexperienced developers.

2. RELATED WORK

Our work focusses on the **onboarding phase** and augments it with **dashboard-like monitors** of testing culture. Here, aspects of social transparency and **gamification and competition** come into play. In this chapter, we give an overview of related work in these fields and shortly consider the newcomer's **educational background**.

The **onboarding** phase is difficult and demanding for the newcomer. Steinmacher et al. present seven categories of barriers that the new hire has to overcome before becoming a fully integrated and valuable employee [18]. For example, 'Problems with Documentation' often hindered new developers with getting started. Practitioners' experience with the testing skills of newcomers, however, deserve particular attention. Begel et al. observed eight new hires at Microsoft during their onboarding phase [3]. Their technical knowledge generally was quite well and they showed a decent will to test — but omitted writing tests due to infrastructural problems. Weyuker et al. report on a company-internal program at AT&T to help new hires in becoming professional software testers [22], which proved costly and longish due to the lack of 'formal testing instruction as part of their academic training'.

Practitioners have noticed a gap between their expectations and the skills that graduates bring to the table — and advocate a more real-world oriented **education** [6]. Practitioners describe university training in software testing as "widely second-rate" [12]. They criticize that most testers are self-taught. Educational institutions, on the other hand, make an effort to convey the role and value of testing to undergraduates — with varying results. Pham et al. report on the testing behavior of soon-to-be Bachelor graduates during a four month software engineering project [15]. A dismissive attitude, lack of technical skills and a high learning curve for testing in a real project kept most students from systematic and automatic testing. Here, a more hands-on approaches to lecturing promise a better adoption of testing. Clark et al. report on the benefits of peer testing [5] in a software engineering project course. The most outstanding benefit was that 'students have seen the necessity of testing', which is a prerequisite for a successful career in software engineering.

Dashboards help the developer to develop and maintain a metal model of a project's progress. Treude et al. give an overview of dashboards that foster awareness of a project's status, developer's activities and artifacts [20]. IBM's Jazz [11] aggregates project status and developer activities in configurable dashboards via so-called viewlets, using the Eclipse infrastructure. Tools like FASTDash [4] and SeeSoft [9] work on a shared codebase to gain and maintain awareness of team members' activities (e.g. who is looking at which file, who edited which file). While these tools aim to increase the awareness for different aspects of the development process, our approach focuses specifically on *awareness and increase of testing activities*. We introduce traits of social translucence directly into the IDE.

Ranking team members and introducing the element of **competition** to inexperienced developers can have beneficial impacts on their development behavior. Dubois et al. propose to research a framework for using gamification in software engineering [8]. They made software development metrics (for example, Test Coverage, #lines of JavaDoc, Code Duplication) of one student software project team visible to another student software project team. Considering these metrics, the team that saw the other team's stats outperformed it. Whether Dubois et al. designed their system to incorporate traits of social transparency is not described. Singer et al. ranked members of a student software engineering project by the number of their commits and rewarded more commits with better badges [17]. This ranking was only visible to the team and led to more frequent and smaller commits, which was desirable. While these experiments show promising results, none of them focus specifically on improving the testing behavior of inexperienced developers.

3. SOCIAL TRANSPARENCY AND SOCIAL TRANSLUCENCE

Making the activity of team members in a computersupported collaboration visible has been subject of much research in CSCW in the past years. Here, we shortly present the two main frameworks of making a collaborator's activity visible: social translucence and social transparency. Our approach uses core concepts of these frameworks and we will relate it to them in more detail in our Discussion, Section 8.

The term *social translucence* was coined by Erickson et al. [10] in 2000. Erickson et al. argue that, in contrast with physical interaction, digital interaction lacks social information and context of communication partners. They propose to make participants and their behavior more visible to one another and define three characteristics that influence how we interact with each other digitally: *visibility, awareness, accountability*. Making social information *visible* gives us the chance to become *aware* of our conversation partner's context and act according to our cultural social rules — otherwise, we could be held *accountable* for our misbehavior. For example, seeing a coworker intently programming,

[2]GitHub.com is an online repository platform with a low barrier for participation. Projects can be browsed by attributes and every user's actions such as commits, discussions, and other artifacts are visible to other users. This facilitates understanding and reproduction of the solutions that other users exerted.

we will not disturb that colleague or we will have to explain ourselves. However, in the digital communication system of a company's chat room application, the coworker's situation may not immediately be comprehensible. Here, the chat status "do not disturb" can convey such social information — and influence our interaction behavior.

Stuart el al. introduce an overarching framework and consistent terminology for the concept of *social transparency* of collaboration in groups or online communities [19]. They define social transparency as availability of social meta-data surrounding information exchange and build upon the concepts of social translucence. However, putting an emphasis on online collaboration of groups and communities, they extend their framework with concepts of authentication, anonymity and provenance. Stuart et al. adapt the concepts of being aware of another collaborator's actions to the characteristics of modern, digital collaborations — such as social networks, global software engineering or social coding platforms. They identify three dimensions of transparency and discuss their first order effects (immediately visible) and possible second order effects (long term effects): *identity, content, and interaction transparency.* Stuart et al. define *identity transparency* as "the visibility of the identity of the sender and/or receiver in an information exchange". Being identifiable is likely to heighten a collaborator's sense of accountability. The origin visibility and history of changes to an information is defined as *content transparency*, where information can also be artifacts produced in a collaboration. A high content transparency lets developers view changes to the artifact and understand its evolvement better by supporting the creation of a mental model. For example, SVN revision history and visualization thereof can help a developer understand the evolvement of a written test suite better and facilitate a narrative of the workflow. Stuart et al. hypothesize that content transparency can increase productivity when creators of artifacts know that their actions are visible. However, they argue that a high content transparency can also increase stress on the creator's side. *Interaction transparency* refers to the visibility of third party access on an information exchange between two partners. For example, how easily a developer can see that other users are interacting with or accessing an artifact, such as a specific readme-file. A high interaction transparency lets users judge popularity of an information exchange and even make inferences about normative behavior within a group. For example, in online communities, newcomers learn from the exchange between senior members and eventually adapt community norms [13].

Social transparency has been subject to much recent research. Dabbish et al. investigate the effect of social transparency on collaboration on the social coding site github.com [7]. In a related study, we investigated the effects of social transparency on the testing behavior of github users [16]. The social transparency traits of social coding sites are currently assessed for use in education [24]. However, social transparency can also have negative effects [14] and social transparent system should be designed with care [19], [10].

In this work, we want to strategically employ traits of social transparency into the work environment of a newcomer. **By introducing so-called testing signals to the IDE, we aim to make team member's testing activity more visible to the newcomer — and increase the newcomer's own testing activity.**

4. RESEARCH PROCESS

As a first step, we seeked to get a better understanding of the nature of testing signals. We re-analyzed the data from our earlier study [16] and studied 15 popular github projects. We looked for signs and artifacts of testing activities that were already present on github's project interface. We curated a preliminary list of testing signals that could be perceived by an novice developer and that could potentially lead to more test writing. These testing signals, mostly artifacts, were visible on github's project interface without much user interaction. The list includes, for example, the aforementioned CI-Badge, obvious test-folder and test-files in the project tree, and documentation about the project's test process.

Next, we focussed on modern IDEs (Eclipse, IntelliJ, MS Visual Studio) and analyzed what testing signals were already present. We took the point of view of an inexperienced software developer — such as new graduates — and studied how testing activities were displayed. All IDEs provided specific views for systematic testing. These views were separate from the main development workflow. In fact, all IDEs allowed for implementing a software project without touching these test views at all. These views are designed to be accessed *when the developer has already decided* on writing automated tests. This, however, is problematic when an inexperienced developer ignores systematic testing.

Our next goal was to design additional testing signals — from any information regarding the team's testing activities that is available in a modern software development collaboration (such as versioning information, authorship information, ...). Again, we took the point of view of an inexperienced developer and employed a GQM approach [2]. Our research question was: "How can the visibility of the team's testing activities be improved, from the point of view of a newcomer?" We used goal trees to break down our goals and get a more fine-grained, concrete understanding of them. Abstraction sheets [1] [21] pushed us to concretize our reasoning behind a new testing signal — and allowed us to back-check our reasoning regularly. Eventually, we produced six testing signals: Currently, we are refining these signals iteratively with the help of multi-staged evaluation rounds (Section 7) with inexperienced developers — in this case, students shortly before receiving their Bachelor's degree and entering today's software engineering workforce. Among originally eight potential signals, these six conveyed the strongest call to test more and were understood best by our population.

5. TESTING SIGNAL REFERENCE MODEL

We present a first, simple reference model for testing signals — *varying forms of visualizing testing activities with the goal to facilitate more testing activities.* This model is derived from our experiences with studying testing signals so far. It describes similarities across different testing signals. We do not claim completeness for this model. We understand that not all testing signals fully feature every aspect of this model — however, we take a pragmatic view on that matter and regard this model as a first iteration. It can help researchers in extending and refining the testing signal approach in a standardized manner. We explain the reference model while introducing our first testing signal 'Latest Test Code Commits' as an example (see Fig. 1).

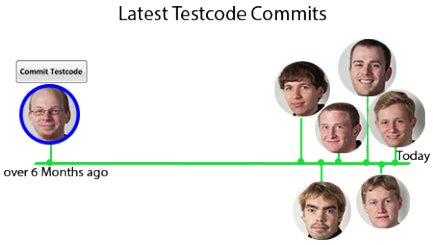

Figure 1: Visualization of the testing signal 'Latest Test Code Commits'. It shows the team's time line of test code committing.

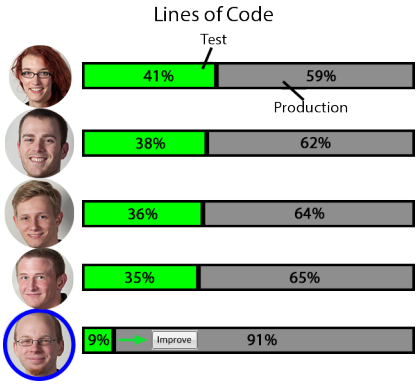

Figure 2: Testing signal 'Test Code Ratio' displays the team's ratio of test code vs. production code.

Strategy of a testing signal: The general goal of testing signals is to facilitate more testing effort in the perceiver of the signal. However, the number of artifacts or actions of team members that can be visualized and can potentially have the desired effect on the perceiver is large. When designing a testing signal, we need to reason about the expected effect of this signal: *why should this specific perspective be visualized as a testing signal and what effect do we expect this to have on the perceiver?* We used a *"If the novice sees that ..., then she will ..."* notation to structure our strategies. For example: If the novice sees that all other team members have committed test code lately, she will commit test code herself.

Mode of a testing signal: Visualizing specific aspects of a team's testing activities can have positive effects that encourage the novice to write more tests. However, it can also have negative effects that keep the novice from testing. Regarding Fig. 1, seeing vivid committing of tests can encourage to write tests on her own (positive effect). However, if most of the team has not committed tests lately, this could communicate to the novice that testing is not important (negative effect). The *mode of a testing signal* describes whether or not the signal can have only positive or both positive and negative effects as well.

Range of a testing signal: When designing a testing signal and its visualization, one needs to consider all of its possible states. This is an important step, as different states of the visualization can have different effects on the novice (different modes of the testing signal). They can take many different forms, such as a limited or unlimited number of discrete states, or a continuos value (for example, the percentage value of tested lines of code vs. untested lines of code). However, not *all* possible states are relevant — subsets can have a similar effect on the novice. Similar to partition testing, we define the states that a visualization can assume and that have a specific effect on the novice as the *range of a testing signal*. In the case of 'Latest Test Code Commits', there are different states to consider: In Fig. 1, the novice can be isolated to the left (encouraging), he can join the herd on the right (encouraging), he can be isolated to the right (encouraging), or he can be joined by the majority of the team on the left (discouraging). [3]

Threshold of a testing signal: Distinguishing between positive and negative (mode) in the set of different states of a visualization (range), the threshold of a testing signal describes the tipping point between the set of positive states and the set of negative states. This threshold depends on the goal of the visualization and is set by the designer of the testing signal. For example, if we want to encourage the novice to test, we should present her with a high testing activity around her — and not with a barely existent testing culture. Regarding the signal in Fig. 1, we set the threshold to "majority of the team is on the right side".

Remedy Strategy of a testing signal: Displaying a novice's negative testing performance through our testing signal visualizations can be discouraging for the novice. We advocate to bundle such visualizations with a fast and low barrier way to remedy the situation. We define this 'way out' as the *remedy strategy of the testing signal*. However, the tools to implement these strategies are beyond the scope of this work and are meant as ideas for further research. When the novice sees that his test commit quota is low compared to his team members, he can use the 'commit test code' button (see Fig. 1). The IDE could jump to a generated test class stub to help the novice commit a new test.

6. TESTING SIGNALS

In this Section, we present visualizations of six testing signals and relate them to our testing signal reference model. Fig. 7 shows all testing signals integrated into Eclipse.

Testing signal 'Test Code Ratio': In our previous study with soon-to-be graduates, one of the inhibiting factors for automated testing was that students felt unproductive when writing tests [15]. They would rather implement more functions and the time they spent on writing test code seemed lost to them. In professional software development, systematic software testing takes up a lot of time and ressources — sometimes as much as developing the production code itself. The testing signal 'Test Code Ratio' is designed to give the novice a sense of how much test code a senior developer writes and what a normal test code vs. production code ratio looks like (see Fig. 2). The strategy of this testing signal is: *'If the novice sees that other team members are writing a substantial amount of test code (and not only source code), too, writing tests code will seem less unproductive to her and she will write more test code'.*

Testing signal 'Test Code Explorer': In our study of testing behavior on GitHub.com, we saw that just seeing test classes around the project nudged contributors to write their own tests [16]. We want to use a similar effect. Often,

[3]There are more states, in which team members are distributed all over the timeline.

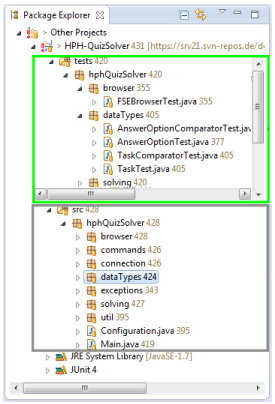

Figure 3: Testing signal 'Test Code Explorer' displays test classes more prominently in the IDE.

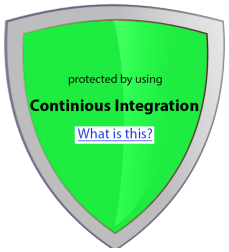

Figure 4: Testing signal 'Using Test Services' displays the use of testing tools in a project and emphasizes the team's commitment to systematic testing.

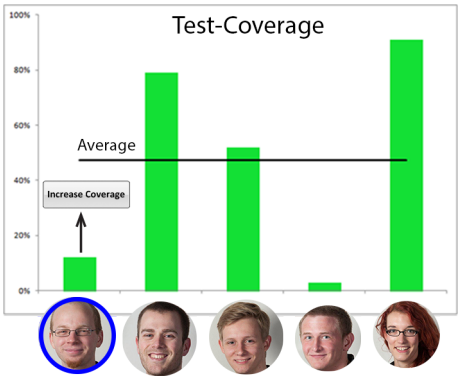

Figure 5: Testing signal 'Test Code Coverage' shows the team's effort to cover their code with tests.

test files are hidden away in the project tree in a rather an easy-to-overlook folder named 'test'. The testing signal Test Code Explorer' places a stronger focus on the test suite and gives the test suite its own prominent placing in the IDE (see Fig. 3). Such a prominent placement lowers the barrier for accessing and adding to the test suite. They underlying strategy of this testing signal is *'If the novice sees test files, she will be reminded to test herself and write more tests'.*

Testing signal 'Using Test Services': Clear signals of test activities and testing tools encouraged code contributors on GitHub.com to add tests to their contributions [16]. In this case, a badge symbol for a continuous integration service on the project's profile side conveyed the team's testing activities. We want to use a similar effect and integrate a stylized shield into the IDE (see Fig. 4). It symbolizes protection and refers to the use of CI in this project. We suspect that novice developers may not be familiar with CI. As a remedy strategy, the testing signal offers a link to a definition and tutorials on how to use it and set it up.

Testing signal 'Test Code Coverage': Similar to the testing signal 'Test Code Ratio' (Fig. 2), the testing signal 'Test code Coverage' compares the novice's testing performance to the performance of the team (Fig. 5). This time, the focus lies less on a healthy ratio but rather on testing performance as measured in test code coverage. This testing signal is more of competitive nature and less focussed on conveying the normative behavior of the team. Code coverage — as a value — is measured by a percentage number, with 100% being the highest. Such a measure is easy to grasp. The strategy for this testing signal is *'If the novice sees that her test coverage is below average, she will try to increase her test coverage by writing more tests'.*

Testing signal 'Test Code Documentation': Good Process and tutorial documentation can be of great help when a developer tries to learn a new technology. Regarding the adoption of testing practices by students, however, there seems to be a shortage of accessible documentation and tutorials. After a four-month software project, students complained that they could not find suitable and accessible tutorials for their testing needs online [15]. Additionally, on social coding sites, easy accessible guidelines are important for communicating a project's testing culture [16]. In the standard view of the popular IDE Eclipse, however, document files are not featured any more prominently than other files. The testing signal 'Test Code Documentation' proposes to grant instructional documentation files a more prominent placement.

7. PRELIMINARY EVALUATION RESULTS

Our work is a first, explorative study in designing and using testing signals. We aim to ensure the applicability of our approach and want to understand its effects. We qualitatively evaluated it regarding the two questions *'What effect do testing signals have on newcomers and do they understand them?'* (understandability) and *'Which testing signal have the highest or lowest effect?'* (ranking). We present preliminary results here.

7.1 Study Design

We conducted a two-stage qualitative evaluation among 24 computer science undergraduates (5th semester or up) at Leibniz Universität Hannover, Germany. These undergraduates are nearly finished with their Bachelor studies and are about to enter the job market. All of them attended the lectures 'Software Engineering' and 19 additionally attended the lecture on 'Software Quality'. It was explained to each participant that she was the newest addition to a team of senior software developers. The other team members were introduced with photographs (as used in the testing signals). At the first stage of our evaluation, the understandability of the graphical visualizations was evaluated. For each testing signal, we interviewed three students independently for about 20 minutes. Each interviewee was presented with a printout of a standard Java Eclipse IDE and included one

Figure 6: Instructional and educational files are placed more prominently by the testing signal 'Test Code Documentation'.

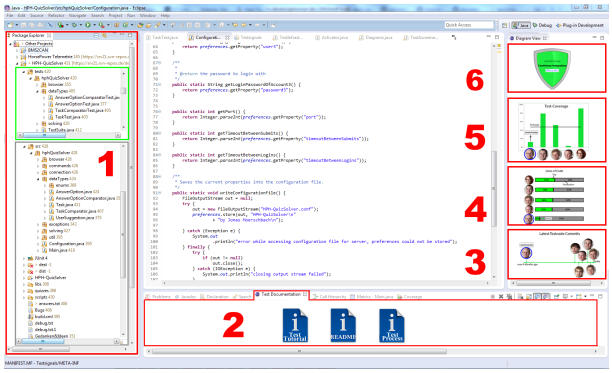

Figure 7: All six testing signals integrated in Eclipse: 'Test Code Explorer' (1)', 'Test Code Documentation' (2), 'Latest Test Code Commits' (3), 'Test Code Ratio' (4), 'Test Code Coverage' (5) and 'Using Test Services' (6)

testing signal as a pen&paper prototype (similar to Fig. 7, with one signal only). The participant was asked to describe the visualization, its message and her emotions when viewing it. We inquired what her next action would be after seeing this visualization. Finally, each participant was invited to give feedback for improving the visualization. The second stage of our evaluation served to determine the testing signals' impact on the graduate. We wanted to understand how to rank the different testing signals regarding their encouragement for testing. We interviewed six students and presented each with all six testing signals (as seen in Fig. 7). As a sanity check, we required each participant to describe all visualizations briefly. This way, we verified that they had understood the message of the testing signals. We asked them to rank the testing signals according to the perceived influence to start writing tests, from high to low. Further, we inquired why they had ordered the testing signals this way.

7.2 First Evaluation: Understandability

Conveyed Message and Impacts on Students. Nearly all testing signals conveyed the message that they were designed to convey. Participants understood that their testing activity was rated as low and — in comparison to their team members' — needed to improve. *"My testing activity is too low. I need to do more."*
Our participants felt motivated to improve themselves. They reported that being ranked to colleagues so visibly gave them an extra nudge to improve and sparked competition. *"I need to write more tests to get a higher rank."*
Using social transparency to make testing behavior explicitly visible — and thus, comparable — can evoke different feelings in novices. Seeing their own test performance made our participants a little bit uneasy: They reported to have a slightly bad conscience for not testing. Most participants did not feel offended — however, three participant's first reaction was indeed to 'click away' the testing signal. They simply did not want to be confronted with their own misbehavior. There is a fine line between rejection and adoption when it comes to personal mistakes or shortcomings. Care needs to be taken when presenting these insights to people and more research is needed to fully understand these effects. On that matter, four students noted that the feeling of uneasiness and the reaction to it are dependent on their relationship to the other team members and the working atmosphere in general.

Improving the Visualizations. The visualizations used in this first evaluation round were first prototypes. We asked for improvements and participants contributed valuable feedback to all visualizations. Feedback mostly consisted of minor improvements like adjusting colors (not red, but more green and blue) or using clickable buttons instead of hyperlinks (signaling the remedy strategy). Figures 1 - 6 show the final visualizations of all testing signals.
Two students had a noteworthy request for the testing signal 'Test Code Explorer' (see Fig. 3). They proposed to place the window containing the productive code above the one containing the test code. Their explanation emphasizes one aspect of the problem that we want to address: *"Productive code is still more important, isn't it?"* One inhibitor to systematic testing in student projects is that students attribute testing little importance and value source code and new function more than quality assurance efforts [15]. However, successful software companies promote a sophisticated software testing culture and award systematic testing the same importance as coding [23]. In this sense, we aim to convey to the novice that testing is of equal importance.

Effects of the Remedy Button. Two third of the students wanted to click on the remedy button — simply *'to see what happens'*. Using labeled buttons combined with arrows successfully conveyed the offer of help. Students correctly assumed that they would get help on how to improve their testing performance, when clicking these buttons. Generally, students wanted to improve — especially regarding in rankings that explicitly compared them to others (testing signals 'Test Code Coverage' or 'Tests Code Commits'). The remedy button served as an easy way to start.

7.3 Second Evaluation: Ranking Impact

We suspected that the different testing signals had varying impacts on the novice developer: some testing signals conveyed the need for testing activity more explicitly while other testing signals were designed to be implicit in nature. We inquired wether our students felt influenced to test more by the testing signals and asked them to rank them from highest to lowest perceived influence. Table 1 shows how often a testing signal was placed at which rank. For example, 'Test Code Coverage' was placed three times in first rank, two times in second rank and once in third rank. Here, we recognized an interesting tendency: Testing signals that used more social transparency features — such as pictures from team members or their activity — were attributed a higher influence by students. They were placed in ranks one to three more often. Meanwhile, more 'passive' and less prompting signals like 'Using Testing Services' and 'Test Code Documentation' were located at the other end of the ranking. The 'Test Code Explorer' stayed in middle ground.
Reasoning about the students' ranking, three arguments stood out: First, better knowledge of and higher familiarity with the metric used by the testing signal lead to better ranking. Students were more familiar with the commonly known metrics of 'Test Code Coverage' and 'Test Code Ratio' (number of lines of code) and accepted these better as performance indicators. *"Test Coverage says most about [my testing activities]."* Second, the profile pictures created a very close

Table 1: Ranking of the testing signals. Accumulated number of placements per rank.

Testing Signal	1st	2nd	3rd	4th	5th	6th
Test Code Coverage	3	2	1			
Test Code Ratio		3	2		1	
Test Code Commits		3	1	1	1	
Test Code Explorer		1	1	2		2
Test Code Doc.			2		3	1
Using Test Services			1	1	1	3

connection and facilitated a competitive feeling between colleagues: *"The direct comparison to colleagues has a strong influence on me."* Third, the other testing signals did not prompt the student for more tests as strongly. This is due to their passive and more informative character. They provide information if clicked or viewed but do not nudge the new hire to do something. *"[The Test Code Explorer] does not prompt me to do anything."*

8. DISCUSSION

Our approach focusses on the inexperienced newcomer to a team of senior developers. The team's testing activity are made visible by introducing traits of social transparency and translucence into the IDE of the novice. Making team member's progress visible to other team members can invoke privacy concerns. In this case, only the newcomer can see the progress of senior developers — and assuming a healthy testing culture, seniors should fare well in the testing signals visualizations. However, we propose to make senior participation an opt-in process and recognize that privacy concerns require further research.

Our approach requires an *active to very active* testing culture in order to encourage newcomers to test more. If no one on the team is testing, all testing signals will have a negative mode and be discouraging. With a good testing culture in place, one could argue that the newcomer picks up testing by herself quickly. We agree that a good testing culture is beneficial for overcoming the testing skill gap in inexperienced developers. However, we think that in order to be able to pick up the testing culture on her own, the inexperienced developer must be able to observe it first. Our approach is an continuation of this relation — enriched with effects of testing culture adoption on social coding sites [16]. Additionally, between an active and a very active testing team, there is a wide range of testing efforts — and teaching testing to an inexperienced developer is costly on the team's side nonetheless (mentoring, lectures, coding camps,...). Here, helping the new hire to adopt the team's testing culture quicker is cost-saving.

We now relate our approach to the frameworks of social translucence and social transparency. At its core, our approach builds upon the concepts of visibility and awareness of actions of other team members. Testing signals are essentially dashboards of other team members' testing efforts and performance — they make the team's testing culture *visible*. This, we hope, makes the novice *aware* of the testing efforts around him. By making the newbie aware of how and how hard senior developers around him test, we aim to induce a change of attitude towards testing and an understanding of testing activities as a norm in this team. Currently, the element of *accountability* is not explicitly included in our approach: Newcomer and senior developers are both informed that the testing stats are only seen by the newcomer. Sharing these testing signals with senior developers could create a performance pressuring situation for the newcomer. Our approach does not hold the newcomer accountable for poor testing performance — however, the newcomer can develop such a feeling herself: Seeing her performance displayed against other's performance can make her feel accountable and in our preliminary evaluations, we have seen a similar effect (bad conscious, slight feeling of uneasiness).

Social transparency was defined with online communities in mind first, and collaboration groups second. We understand it as a continuation of the pre-defined framework of social translucence, which coined exchange of social salient information when online collaboration was not as available as it is today. However, our approach also touches on concepts of social transparency — and in fact, was inspired by social transparency effects on social coding sites [16]. Our aim is to recreate these effects in a company work setting: Testing signals use real profile pictures of colleagues. This form of *identity transparency* creates a closer connection to the newcomer and gives the visualizations a bigger impact. Informing the newcomer about how many tests a team member has written or when a team member has written them, brings by a form of *content transparency* for the test suite: The newcomer can maintain a mental model of changes to the test suite and understand its evolvement better. Additionally, seeing who committed tests can help the newbie in identifying expert testers for further questions or advice.

Our evaluations are exploratory and qualitative in nature. The population is small (24 students) and we do not claim statistical significance or generalizability. However, we deem students just before their graduation a suitable evaluation population for our actual target population: graduates that have just entered the job market. We do not think that the experience level in graduates between university and their first day at work differ much. Although results of our two evaluations are promising, we are missing a confirming evaluation that shows a measurable advantage of our approach in a real-world setting. This final evaluation is currently in preparation.

We recognize that displaying all six testing signals at once can seem overwhelming or intimidating, even. The effects of different combinations of testing signals or disabling certain signals still needs research. However, we have made a first step in this direction by evaluating the impact rating of testing signals — less influencing or signals with similar message could be strategically disabled for a less cluttered work environment. We are aware that not all testing signals display positive messages at the same time. Also, testing signals react differently depending on the extent of testing activity. For example, regarding the testing signal 'Test Code Commits' (see Fig. 1): Having committed one little, simple test, will bring the novice to the right, joining the crowd — this may have an uplifting effect, but actually does not imply a heightened and meaningful testing activity. In this case, other testing signals like 'Test Code Coverage' (see Fig. 5) would still display a negative novice performance — test code coverage would not be increased by much. Currently, we are exploring and experimenting with a so-called *smart display* strategy that strategically enables and disables testing signals to increase positive impact on the novice.

9. CONCLUSION

This work introduces the concept of testing signals: dashboard-like visualizations of other team member's testing efforts, integrated into the IDE of an inexperienced newcomer — such as graduate students joining the software engineering workforce. Newcomers sometimes have difficulties with systematic and automatic testing, often this is induced by a dismissive attitude towards testing. By making the team's testing culture visible, we aim to improve adoption of testing activities, resulting in a more efficient onboarding phase. We propose to strategically use social transparency to make testing activity visible and the newcomer aware of it.
We present a first reference model for testing signals and introduce six testing signals. We evaluated our work-in-progress with 24 soon-to-be bachelor graduates with encouraging results: Being reminded of their lagging test progress induced the need to test more. Visual comparison to colleagues' testing performance woke competitive feelings in students and made them want to write more test. Currently, we are preparing a final evaluation to validate a measurable advantage of our approach for onboarding an inexperienced developer.

10. REFERENCES

[1] V. R. Basili. *Software modeling and measurement: the goal/question/metric paradigm*. Computer science techn. report Series. Maryland Univ., 1992.

[2] V. R. Basili and D. M. Weiss. A methodology for collecting valid software engineering data. *Software Engineering, IEEE Transactions on*, SE-10(6):728 –738, nov. 1984.

[3] A. Begel and B. Simon. Struggles of new college graduates in their first software development job. In *ACM SIGCSE Bulletin*, volume 40, pages 226–230. ACM, 2008.

[4] J. T. Biehl, M. Czerwinski, G. Smith, and G. G. Robertson. Fastdash: a visual dashboard for fostering awareness in software teams. In *Human Factors in Computing Systems, SIGCHI*, pages 1313–1322, 2007.

[5] N. Clark. Peer testing in software engineering projects. In *Sixth Australasian Conference on Computing, ACE*, pages 41–48, 2004.

[6] R. Conn. Developing software engineers at the c-130j software factory. *Software, IEEE*, 19:28–29, 2002.

[7] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Computer Supported Cooperative Work, CSCW*, pages 1277–1286. ACM, 2012.

[8] D. J. Dubois and G. Tamburrelli. Understanding gamification mechanisms for software development. In *9th Joint Meeting on Foundations of Software Engineering, ESEC/FSE*, pages 659–662, 2013.

[9] S. G. Eick, J. L. Steffen, and E. E. Sumner, Jr. Seesoft - a tool for visualizing line oriented software statistics. In *Software Engineering, IEEE Transactions on*, pages 957–968, 1992.

[10] T. Erickson and W. A. Kellogg. Social translucence: an approach to designing systems that support social processes. *ACM transactions on computer-human interaction (TOCHI)*, 7(1):59–83, 2000.

[11] R. Frost. Jazz and the eclipse way of collaboration. *Software, IEEE*, pages 114–117, 2007.

[12] R. L. Glass, R. Collard, A. Bertolino, J. Bach, and C. Kaner. Software testing and industry needs. *IEEE Software*, 23(4):55–57, 2006.

[13] R. Kraut, M. Burke, J. Riedl, and P. Resnick. Dealing with newcomers. *Evidencebased Social Design Mining the Social Sciences to Build Online Communities*, 1:42, 2010.

[14] D. T. Nguyen, L. A. Dabbish, and S. Kiesler. The perverse effects of social transparency on online advice taking. In *18th Conference on Computer Supported Cooperative Work & Social Computing, CSCW*, pages 207–217. ACM, 2015.

[15] R. Pham, S. Kiesling, O. Liskin, L. Singer, and K. Schneider. Enablers, Inhibitors, and Perceptions of Testing in Novice Software Teams. In *22th Foundations of Software Engineering, FSE*, 2014.

[16] R. Pham, L. Singer, O. Liskin, F. Figueira Filho, and K. Schneider. Creating a shared understanding of testing culture on a social coding site. In *Int. Conf. on Software Engineering (ICSE)*, pages 112–121. IEEE Press, 2013.

[17] L. Singer and K. Schneider. It was a bit of a race: Gamification of version control. In *2012 2nd International Workshop on Games and Software Engineering (GAS)*, pages 5–8, 2012.

[18] I. Steinmacher, I. S. Wiese, T. Conte, M. A. Gerosa, and D. Redmiles. The hard life of open source software project newcomers. In *Proc. of the 7th Int. Workshop on Cooperative and Human Aspects of Software Engineering*, pages 72–78. ACM, 2014.

[19] H. C. Stuart, L. Dabbish, S. Kiesler, P. Kinnaird, and R. Kang. Social transparency in networked information exchange: a theoretical framework. In *Computer Supported Cooperative Work, CSCW*, pages 451–460. ACM, 2012.

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

[21] F. Van Latum, R. Van Solingen, M. Oivo, B. Hoisl, D. Rombach, and G. Ruhe. Adopting gqm based measurement in an industrial environment. *Software, IEEE*, 15(1):78 –86, jan/feb 1998.

[22] E. J. Weyuker, T. J. Ostrand, J. Brophy, and R. Prasad. Clearing a career path for software testers. *Software, IEEE*, 17:76–82, 2000.

[23] J. A. Whittaker, J. Arbon, and J. Carollo. *How Google tests software*. Addison-Wesley, 2012.

[24] A. Zagalsky, J. Feliciano, M.-A. Storey, Y. Zhao, and W. Wang. The emergence of github as a collaborative platform for education. 2015.

Towards GEEZMO: hiGh-frEquEncy Zest and Mood-pOlling for Proactive Software Development Problem-Solving

Elisabetta Di Nitto, Raffaela Mirandola, Santi Raffa, and Damian A. Tamburri
Politecnico di Milano
Via Golgi 42
Milan, Italy
{elisabetta.dinitto, raffaela.mirandola, damianandrew.tamburri}@polimi.it, santi.raffa@mail.polimi.it

ABSTRACT

Development of software is happening on an increasingly distributed fashion. Individuals normally coordinate and interact over any combination of IRC rooms, mailing lists, private email, etc. Conversely, big players like Google Inc. employ yearly Googlegeist polls asking its employees how they feel about the company, its directions and its managers. As a consequence, the amount of time required for managers and management to feel the pulse of their subordinates is removed from actual work. We argue that software development status can be computed by eliciting just a few bits of information that can be anonymously extracted by a single poll - the poll can be completed very quickly and its administration can happen as a simple prompt inside windows that developers would be visiting anyway (e.g., the log-in screen, commit screens, etc.). This paper elaborates this idea and develops a prototype to articulate the idea in practice. Finally, the idea is discussed using a preliminary validation by means of statistical methods and simulations.

Categories and Subject Descriptors

K.6.1 [**Project and People Management**]:

General Terms

Theory

Keywords

Mood-Sensing, Social Software Engineering

1. INTRODUCTION

Development of software is happening on an increasingly distributed fashion [10]. On one hand, open-source software development happens almost invariably on multiple locations, involving individuals who do not know each other other than by their screen names and email addresses. Individuals coordinate and interact over any combination of IRC rooms, mailing lists, private email, bug trackers,etc.

SSE'15, September 1, 2015, Bergamo, Italy
© 2015 ACM. 978-1-4503-3818-9/15/09...$15.00
http://dx.doi.org/10.1145/2804381.2804383

On the other hand, closed source software development is following this trend, e.g., by means of Global Software Engineering (GSE) [22]. As a notable example, Stack Exchange Inc. is the company behind a popular, closed source Q&A system. Their developers are distributed across multiple continents [5]; the company relies on a proprietary chat system, Google Hangouts and the online bug-tracking system Fogbugz for coordination. While some of the core components are made open-source [6], most of the development happens behind closed doors.

As a result, communication between team members happens at the same time over multiple communication channels of several different types and goals, ranging from written and formal communication (mailing lists, bug trackers) [23].

The main challenge here is that most of those systems cannot be adequately monitored or data-mined, either for technical or administrative reasons. A manager wishing to proactively diffuse situations before they detonate would have a difficult time being in the right place at the right time. They may read IRC logs, or parse them using so-called opinion mining [18], but it is unfeasible to do so on every system the team may use for lack of adequate APIs and the possibility of private communication being legitimately unavailable to a manager due to expectations of privacy.

As a result, managers usually rely on regular one-on-one meetings with their developers, e.g., in order to be aware of any organisational and socio-technical issues [21] as they may come up. Additionally, companies invest significant resources in infrequent employee satisfaction surveys in order to read their workforce morale; for example, Google Inc. employs yearly Googlegeist polls [2] asking its employees how they feel about the company, its directions and its managers.

Any amount of time required for managers and management to feel the pulse of their subordinates is removed from actual work. As a result, there is a very clear trade-off to be made between polling "bandwidth" and "frequency": running one-on-one meetings on a daily basis would be a questionable use of company time, while administering long opinion polls too frequently might result in understandably decreased response rates and increasing poll satisficing [14].

We find that this trade-off between time resolution and data resolution (curiously reminiscent of the Heisenberg uncertainty principle [9]) is usually resolved in favour of data resolution; however, another physical principle that may apply here is the Nyquist rate [11]: employee sentiment can vary very dramatically very quickly, and situations may need to be handled within hours of their insurgence. This can become quite complicated when managers have to attend off-

site meetings at times, as they are usually requested. Conflict resolution (if any) is then delegated to the manager of the manager, who may not be in touch with a team's interpersonal relationships; in order to enable them to do so (in addition with manage their managers), they usually have to organise one-on-one meetings of their own.

What if we tried a different trade-off?

This paper proposes GEEZMO, i.e., hiGh-frEquEncy Zest and Mood-pOlling for software development success. By elaborating on GEEZMO, this paper tries to explore the opposite end of the previously mentioned time-data trade-off spectrum: what if project status updates are obtained by asking very little information very frequently? In particular, we propose the use of a daily one-question poll: "How was your day?" The response can be chosen between only a few values (for example, five values ranging from "amazing" to "terrible"). These bits and pieces of information intuitively amount to the "zest" of software developers, i.e., their engagement and energy towards their development scenario and problems as well as their "project mood", i.e., their current status of organisational and social tranquillity they might perceive in that scenario.

On one hand, only a few bits of information can be anonymously extracted by a single poll. On the other hand, this means the poll can be completed very quickly and its administration can happen as a simple prompt inside compatible high-traffic online services you would be visiting anyway. Additionally, since we would not be submitting just one poll, but one poll every day, we gain a lot of implicit information by contextualising the responses with their time frame: spikes in mood might be expected right after a company-sponsored off-site social event, for example; crunching, on the other hand, naturally lowers the developers' quality of life and thus their morale. If you only poll every 12 months, you only can see the aggregate result of one year's worth of events.

This paper illustrates our first attempt at proposing a solution to follow this research path. Finally, we offer a preliminary validation using statistical methods and simulations of a prototype.

2. RESEARCH PROBLEM AND PROPOSED SOLUTION

The research problem we want to tackle concerns how to come up with reasonable estimation of project status using reported many-eyes mood and zest across a software project. Our research question can be phrased as follows: "what project status updates can be obtained by asking very little information very frequently?". The layout of our simple, non-invasive research solution is shown in Fig. 1.

In the following we elaborate the fundamental assumptions and design principles behind GEEZMO, i.e., the solution we propose in pursuit of said research question.

GEEZMO distinguishes between three roles, each corresponding to successive levels in a corporate hierarchy:

- the "voters", people the morale of which we want to monitor;
- the "predictor", a person who is in charge of keeping the morale high;
- the "supervisor", the predictor's manager.

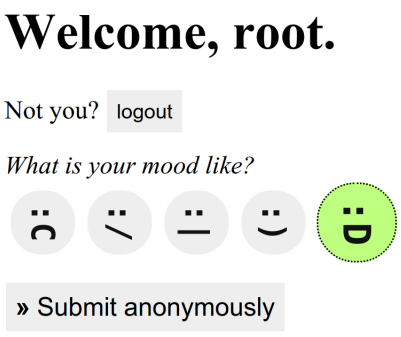

Figure 1: GEEZMO, a Splash-Screen Visual.

GEEZMO asks the voters about their morale, and we shall call that response "vote" by extension. At the same time, it asks the predictor to make an educated guess of his voters' aggregate response; we will similarly dub the predictor's response "prediction".

Distinguishing between votes and predictions allows elaborating possible scenarios:

1. If the two match and they are high, everything is fine.
2. If the two match but they are low, the manager is likely to know what's the issue and we can assume he's working on fixing the issue. There is no cause for further alarm.
3. If the two differ, however, we have a situation that is not being dealt with: the manager's perception of reality does not match what's actually going on. If the situation persists, we warn the supervisor.

Based on these scenarios we are thus looking for a solution that must:

1. Measure the mood of a (relatively) time-independent small population on a short but regular interval;
2. Measure the teams' own awareness of that mood;
3. Raise an alarm when the two diverge;
4. Do so in a time-effective manner, so to minimize the efficiency cost of such a system;
5. Do so in a way that is not self-defeating, that is, the use of GEEZMO shall not accidentally or indirectly punish users for their honesty or pressure them to vote in a given way;
6. Do so in a way that does not cause unnecessary drama, especially when false alarms occur.

In matching the above goals, we elaborated the following design principles:

No pressure. GEEZMO shall not introduce pressure on people to vote a particular way. Particular case shall be added in managing "feedback loops" that result in pressure from outside. For example, by granting anonymity to workers, we seek to reduce consequence for speaking your heart out when filling in the form.

No distraction. GEEZMO shall strive to minimize the cost of doing a mental context switch from actually getting things done to feeding data into GEEZMO.

No miracles. A piece of software that asks a single question and gets a single response can actually do much, but no more than that. Human intervention is still necessary to fix the kind of problems GEEZMO detects and we, indeed, trust a worker's supervisor to do the right thing.

No frills. GEEZMO shall be made available in the easiest form possible, e.g., though the machine log-in or IDE access screen. Additional features within GEEZMO would be unnecessary cognitive burdens.

No trust. Users shall not have to blindly trust GEEZMO or their managers in order to have the peace of mind to vote normally. Data that shall not be disclosed shall not be stored to begin with. GEEZMO shall be resilient against (or at least allow detection of) attempts to sabotage through artificial voting patterns.

No forcing. GEEZMO shall make no assumptions the organisational social structure in the organisation in question, e.g., if it is hierarchical (typical in closed-source) or circular (typical in open-source). In so doing GEEZMO, must be designed both high-level organisational scenarios, namely, hierarchical and circular. Although we tested a preliminary application of this principle in our current prototype, our results suggest it may need further research. More detailed discussion can be found in Section 5.1.

In applying the principles above, we developed an open-source, proof-of-concept implementation written in Django (version 1.6).

The implementation reuses Django built-in standard components such as users, authentication mechanisms, signals, models and templates in an attempt to be as modular as possible. This should ease future deployment into arbitrary environments; for example, replacing Django's built-in authentication back-end with another one requires you to only write a couple of methods, whereas plugging in a different template system is only a matter of importing its Python 2.x bindings from `views.py` and replacing the method calls as needed.

The git repository for this implementation of our proof-of-concept prototype can be found online[1].

3. RAISING ALARMS

We now have devised two different schemes for determining when a prediction is "successful", but that only gets us so far. We also need to decide when enough non-successful predictions become a probable cause for alarm.

If we use 2 as the number of unsuccessful predictions triggering an alarm for hierarchical teams and 5 for circular teams (where all voters can also be predictors), we can consider each day's predictions as respectively a Bernoulli trial with $p = 0.5$ (as calculated in table 1) and a Bernoulli series with $p = 0.4$.

[1] `http://github.com/badp/weather`

Let us consider the number of successful votes s out of the total number of predictions m: we expect $s \simeq pm$. Intuitively, if $s \ll pm$, then there is unexpected unrest, whereas if $pm \ll s \simeq m$, chances are the system is failing to acquire meaningful data from the polling. The Neyman-Pearson lemma [16] confirms that, if we want to test that the maximum likelihood success rate for a binomial series is $\theta = p$ against $\theta \neq p$, the most powerful likelihood-ratio test of size α succeeds if $k_1 < s < k_2$ [4]; in order to derive k_1 and k_2 we can calculate this test's p-value.

In other words, if we are willing to tolerate at most a probability α that the system incorrectly raises an alarm, then the check that can possibly raise the least amount of false alarms succeeds if:

$$P(S = s) = \binom{s}{m} p^s (1-p)^{m-s} \leq \alpha$$

where $S \sim \mathbf{Bi}(m, p)$ is the expected probability distribution of s.

If we consider $t = 5$ to be the number of days the system should take before raising an alarm and $\alpha = 0.05$, we have:

- In the hierarchical case, $m = 5$, $p = \frac{1}{2} = 1 - p$, it follows that $P(S = s) = \binom{s}{5}\frac{1}{32} = \begin{cases} 1/32 & s \in \{0, 5\} \\ 5/32 & s \in \{1, 4\} \\ 10/32 & s \in \{2, 3\} \end{cases}$.
  Thus, we raise an alarm if the number of successes is either 0 or 5.

- In the circular case, we have to apply the formula directly.

  For example, consider a team of $n = 6$ people with $m = 24$ predictions in the last 5 days. Since we are using "five options," $p = 0.4$ and $S \sim \mathbf{Bi}(24, 4)$. By calculating $P(S = s)$ for $s \in \{0, \ldots, 24\}$ we realize that a warning should be raised in case $s \leq 5 \vee s \geq 14$ ($E[S] = mp = 9.6$). Of course, in practice, it is not necessary to calculate all 16 possible values of s: it suffices to compare $P(S = s)$.

Again, in the calculation of these formulas we have made the inaccurate assumption that each prediction's success is independent from other predictions; this is especially a problem in the circular scenario. Future work should seek to take this into account, for example by adopting a suitable approximation of the generalized linear model [15].

As you might notice, GEEZMO has not one but *two* Tables of votes: `Vote` and `TodayVote`. We assume that the `Vote` Table is public, whereas the `TodayVote` Table is private and, in real-life deployments, access to the Table is limited to GEEZMO and only GEEZMO. Votes are added to `TodayVote` and, when midnight rolls, a task processes the contents of the Table. For example, if a team only receives two or less votes in a day, we discard those votes. The order of those votes is also shuffled. The main advantage this measure brings is the guarantee that the current day's votes will not be accidentally leaked by any user-facing views (such as the supervisor's) reading off the `Votes` Table. This measure also allows us to avoid adding a `has_voted_today` flag to the `UserProfile` Table.

The second decision is using two Tables for users: `User` and `UserProfile`. This is mainly a best-practice workaround

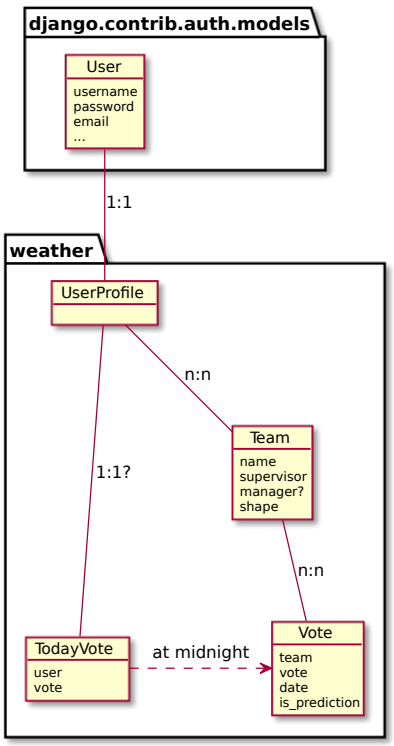

Figure 2: Entity-relation graph for GEEZMO.

on the fact that the `User` Table is provided by the authentication framework, and it is best left untouched. Any additional fields are to be added to `UserProfile` instead. At this point we did not actually identify any suitable additional field to attach to the profile (remember: "no frills"); the class is, nonetheless, a convenient place to store relevant methods. Incidentally, the recommendation to avoid modifying objects from the authentication framework is why GEEZMO does not use Django's implementation of user groups.

As a final remark, when GEEZMO decides to contact the manager, it sends a custom signal to `weather.problem_detected`. The code base provides a simple signal handler that sends an email to the supervisor; additional actions can be defined by creating and connecting new signal handlers in `signals.py`.

Whether we decide to raise an alarm or not, we should support the supervisor in helping decide whether to escalate the situation by starting talks with his manager. We need to anonymise data as much as possible to protect the anonymity of voters. Voters anonymity however can be turned against the system, through simple frequency analysis – and much more can be done to limit the negative effects thereof, for example through "fuzzing" techniques.

Frequency analysis however can also be a force for good, however, as it can uncover some use of anomalous voting patterns such as hostile voting. As a result the supervisor's reporting interface should show not simply one day's voting average and standard deviation, but actual separate counts for each possible vote or prediction. This should prove as an important data point for those evaluating the effectiveness of GEEZMO at measuring team morale in the real world.

Table 1: Estimated likelihood of a prediction "successfully" guessing a single vote when using a "distance from average" criterion [13].

i	$\alpha = 1.5$		$\alpha = 1$	
	r_i	$P(v_i \in r_i)$	r_i	$P(v_i \in r_i)$
1	$(1\ \ , 2.5)$	29%	$(1, 2)$	14%
2	$(1\ \ , 3.5)$	67%	$(1, 3)$	48%
3	$(1.5, 4.5)$	86%	$(2, 4)$	68%
4	$(2.5, 5\ \ )$	67%	$(3, 5)$	48%
5	$(3.5, 5\ \ )$	29%	$(4, 5)$	14%

4. PRELIMINARY VALIDATION: ARE PREDICTIONS "SUCCESSFUL"?

To get an idea for the validity of GEEZMO in practice we used statistical analysis and simulation of a development network composed of 100 voters. This section outlines details of our statistical analysis and the considerations we used for the evaluation of GEEZMO.

Let n voters cast their votes $v_i \in \{1, \dots, 5\}$ anonymously, with 1 being the most negative response and 5 being the most positive. Predictors can also choose $p_j \in \{1, \dots, 5\}$.

We need to define somehow when a prediction p_j is sufficiently close to that day's votes $\mathbf{v}$ to be considered "successful" – and more importantly when it is not and there is cause for concern. Let us consider three approaches to achieve this goal described in the following.

4.1 Distance from Average

Results for this approach are shown on Table 1. Let $\bar{v}$ be the average of votes. With this straightforward approach, the j-th predictor is successful if $|p_j - \bar{v}| < \alpha$, for some value of α that is lax enough to reduce false alarms (for example $\alpha > 1$) but tight enough to allow the possibility of failure (for example $\alpha < 2$). For example, it might seem reasonable to pick $\alpha = 1.5$.

Let us call the range of values $r_j = \left(r_j^-, r_j^+\right)$ of $\bar{v}$ that makes a choice p_j of prediction successful the prediction's "range". This method is not very effective because, effectively, $r_5 \subset r_4$. Indeed, since we had chosen $\alpha < 1$, $r_5^- = 5 - \alpha < 4$ and $r_4^+ = 4 + \alpha > 5$. Perhaps more concerningly, r_3 covers almost the entirety of the possible range of values for $\bar{v}$: if $\alpha = 1.5$, then $r_3 = (1.5, 4.5)$.

Assume, for clarity's sake, that $v_i \simeq \mathcal{N}(3, 1)$[2] are independent (although they clearly are not) and are identically distributed. Even with a "restrictive" choice of $\alpha = 1$, $P(v_i \in r_3) = P(2 < v_i < 4) \simeq 68\%$; with the original choice of $\alpha = 1.5$,$P(v_i \in r_3) > 86\%$. These odds are unacceptably high, especially when compared to the odds of the other choices.

Things get only worse when we compare the predictions against the actual average $\bar{v} = \frac{1}{N}\sum_i v_i \simeq \mathcal{N}\left(3, \frac{1}{N}\right)$: as the number of participants increases, the variance of $\bar{v}$ decreases and thus the likelihood of $p_j = 3$ being classified as successful grows even further.

Of course, v_i's are not independent. Given two jointly normally distributed random variables $X, Y \simeq \mathcal{N}\left(\mu, \sigma^2\right)$ with correlation ρ, it can be shown that $\frac{X+Y}{2} \simeq \mathcal{N}\left(\mu, \frac{\sigma^2(1+\rho)}{2}\right)$;

[2] $v \simeq \mathcal{N}\left(\mu, \sigma^2\right) \implies P(v \in [\mu - 2\sigma, \mu + 2\sigma]) > 95\%$. By choosing $\mu = 3$ and $\mu - 2\sigma = 1$, the choice of $\sigma = 1$ follows.

since we expect $0 > \rho > 1$, the combined variance increases and the situation is somewhat ameliorated. This is but a pyrrhic victory, unfortunately: in the best case scenario, $\rho = 1$, we have $X \equiv Y \equiv \frac{X+Y}{2} \simeq \mathcal{N}(\mu, \sigma)$, and successfully predicting the average of two votes becomes no harder than predicting a single vote.

4.2 Segmented Average

A much better approach could be obtained working backwards from the previous analysis: what if we made each range r_i equally likely? For example, we could consider this simple "two options" rule: the j-th predictor is successful if $p_j = \{1, 2, 3\} \wedge \text{avg}(\mathbf{v}) < 3$ ("bad day") or if $p_j \in \{4, 5\} \wedge \text{avg}(\mathbf{v}) > 3$ ("good day"). It is pretty obvious here that, assuming the initial assumption $v_i \simeq \mathcal{N}(3, 1)$ holds true, random choice will not result in consistent success.

Since the ranges in "two options" are however very wide, we can also consider what happens when we divide the 1-5 range in more equiprobable ranges, separated by the 20th, 40th, 60th and 80th percentiles of $\mathcal{N}(3, 1)$. The results of doing so are shown in approximate form in Table table 2 under the "five options" and "four options" column. In "five options" there is no overlap between ranges and each is theoretically 20% likely to be the correct one. "Four options" obtains its four ranges through pairwise concatenation of ranges from "five options", obtaining overlapping ranges each 40% likely to be correct; a fifth "do not know" option is also included.

4.3 Poll Satisficing and the Midpoint Option

Poll theory points to the phenomenon of poll satisficing [14]: when given a bipolar scale rating with a midpoint, such as in this case, the gives the appearance of being "the easy way out:" the middle option has been shown to be the easiest to defend and, while it takes little cognitive load to choose it, it also gives the impression of careful choice.

Since $3 \in r_3$ for all methods considered so far, picking this option essentially means "this was an average day, neither good or bad." This is probably true of most days, however. If we deny the midpoint, as it was done in the "bad day"/"good day" example, we force predictors to take sides.

As an aside, this might rise the question: why offer the midpoint to voters, then? Would not voters also mindlessly pick this middle option? This seems likely. However, we must not lose sight of the greater context here: voters are developers and their job description is implementing software, not voting on polls. One of the design goals here is to minimize the context switch penalty that daily polls may introduce. Predictors, on the other hand, are managers who are supposed to keep up with and facilitate their team's operations. Making their polling work slightly harder and forcing them to take sides does not therefore seem unfairly taxing.

That does not mean that we should not anticipate and compensate for this effect. One way of doing so is through the option labels: we choose "`:-|`" for $v_i = 3$, which has (in the author's opinion) a slightly negative connotation; an "average" state of mind would rather be described through the smiley face "`:-)`", chosen for option 4. Another way we can account for it is by adjusting the segments further.

This phenomenon is the rationale behind devising the "four options" approach. The midpoint prediction option is still available in the interface, but it is interpreted as a "do not know" response and is not registered. This discards attempts at mindless poll satisficing. Additionally, adjacent intervals overlap, granting predictors a little "grace" interval and reducing the chance of false negatives. Finally, random choice of p_j fares worse than a coin flip.

This analysis might superficially make "four options" the clear winner. However, there are a number of other important issues with it that we will now discuss.

Table 3: An example of GEEZMO configured to support hierarchical structures recursively.

Team	Predictor	Voters	Supervisor
1	C	D, E	B
2	G	H, I	B
3	B	C, F, G	A
4	J	K, L	A
5	B, J, A		A

(Teams 1–4 are hierarchical. Team 5 is circular.)

5. DISCUSSION

5.1 Beyond Hierarchical Social Structures

The presence of hierarchy might be taken for granted in some, but not all, social structures found in software development. Our previous work [24, 23], identifies 13 different types of organizational social structures commonly found in the wild.

In particular, some community types do not have sufficiently strong hierarchies to support classical problem reporting and solving approaches, for example:

- Informal Communities
- Problem Solving Communities
- Informal Networks

These communities present strong egalitarian characteristics and thrive without a strategic hierarchy for typical mood-sensing strategies to exploit. To address this limitation a variant on the "hierarchical team" model in our research solution was be the "circular team" model, where all voters can also be predictors. The consequent challenge we still face is the need to identify one team owner who will act as the supervisor, but otherwise every member act as both somebody who submits information about their morale and somebody who tries to maintain others' morale high. Table 3 recaps organisational social structure types arranged by circular or hierarchical logic.

5.2 Hostile Voting

In our simulations we observed that hostile voting can have at least a 40% chance of changing the predictions outcome – and if the outcome does change, it does in the direction of the hostile voter's random choice of 1 or 5 for that day. Here we see the reverse outcome from the previous analysis: two options virtually guarantees the hostile voter a successful prediction, whereas "five options" and "$\alpha = 1$" perform significantly better. In particular, when comparing their measured 36% success rate with the respective theoretical success rates of 40% and 14%, "five options" is the only solution that performs acceptably.

Table 2: Approximate $\bar{v}$ ranges of "successful" predictions derived from the "segmented average" approach [13].

i	Two options		Five options		Four options	
	$P(v_i \in r_i)$	r_i	$P(v_i \in r_i)$	r_i	$P(v_i \in r_i)$	r_i
1	50%	(1, 3)	20%	(1 , 2.2)	40%	(1 , 2.8)
2			20%	(2.2, 2.8)	40%	(2.2, 3.2)
3			20%	(2.8, 3.2)	(no vote)	
4	50%	(3, 5)	20%	(3.2, 3.8)	40%	(2.8, 3.8)
5			20%	(3.8, 5)	40%	(3.2, 5.0)

Things get even more interesting when more than one people adopt this strategy, however, and this is where "five options" really shines. As game theory tells us, if a strategy is optimal, *all* rational players will use it. If we were to apply gamification to such a system, this would give further incentive to reap gamification's rewards as opposed to truthfully answering the question.

In order to analyze this, we ran another simulation: we analyzed what happens when 100 people vote and an increasing number h of them engages in hostile voting, whereas the rest votes like $\mathcal{N}(3, 1)$. "Two options" unequivocally rewards hostile voting: even as everybody rolls their 1-or-5 vote independently, the chances of getting a correct predictions increase beyond 50% as more people engage in hostile voting[3]. On the other hand, the other methods give additional hostile voters diminishing returns: in particular, you are always more likely to "get it right" if you vote honestly, and "five options" again shines as the solution that performs the best.

5.3 "Everything is Terrible" (or Perfect)

That said, that does not mean that there are no very simple strategy that leads to every voter/predictor being 100% correct 100% of the time. For example, everybody could agree to always pick 1 (a strategy we affectionately call "everything is terrible"). Similarly, everybody could agree to say "everything is perfect" and always pick 5.

There can be no real solution to this without external intervention (from reprimanding the relevant teams to simply giving up). The best we can do is not to encourage people (even if inadvertently) to adopt such a strategy in the long run. This is the main reason why we conjecture that applying gamification to GEEZMO could be counterproductive. More discussions on this conjecture are provided in the next Section.

5.4 Gamification with GEEZMO

Gamification techniques are based in creating a reward and incentive structure in the use of the software. Made originally popular by Stack Overflow, gamification spread quickly to other "social" software with varying degrees of success. The main issue when designing such a system is making it so that attempting to "game the system" in order to reap as many benefits from the system as quickly as possible pushes the users towards desirable behaviour.

For example, Stack Overflow uses two different benefit systems: reputation, which is awarded by other users for your performance, and badges, which are usually awarded by the site for performing a task for the first time (tasks such as "filling out your profile" or "voting an answer down – any answer").

The three main (and essentially only) interactions GEEZMO has are voting, predicting and supervising. We now present a number of badges we could award for each, and why implementing them would be a bad idea:

- "First vote!" (voted for the first time)

  Awarding this badge hampers the anonymity characteristics of GEEZMO. If you gain access to the list of anonymized votes and compare the vote history before and after this badge was awarded, it is possible for a third party to extrapolate how the newcomer is voting. This could also encourage the newcomer to vote something – *anything* – although chances are that a new user would not be savvy to a reward structure built into the website they just used for the first time.

- "Returning customer" (voted n times)

  If, however, the user is awarded a badge with great fanfare for voting for the first time, chances are they will look at the list of badges, and "vote n times"-style badges do nothing but get the user to vote on something – *anything* – for a number of days. This is especially problematic if the badge requires voting for n days in a row. What we would like to have is people voting honestly. What this badge risks actively encouraging is mindless voting for the midpoint.

  Worse still: in order to actually implement such a badge, you'd have to maintain at the very least a counter in your database saying how many votes each user has cast. This gives an attacker more information when performing frequency analysis.

- "Oh, happy days!" (voted 5 for the first time), "I need a hug" (voted 1 for the first time), "Jack of all Moods" (voted for each mood at least once), etc.

  Again, such badges provide unacceptable leaks in anonymity and very strong information in backwards engineering of one user's voting patterns in case those badges have not been awarded.

- "Observer" (predicted for the first time)

  In the hierarchical model of teams, where there is exactly one predictor – the manager – predictions are not

[3]This is perhaps counter-intuitive. If 100 voters/predictors engage in hostile voting using the two options system, the system degenerates in a game where 100 people flip a coin at the same time and the result that is more common in each "round" is also the winning one. If there are 50 tails and 50 heads, everybody wins. If there are 49 heads, there are 51 tails, and thus 51 winners – and viceversa. As a result, your winning chances in this game are strictly greater than 50%.

anonymous and he or she has not much they can do to affect the outcome of voting, short of flat out telling people what to pick. In a way, there is no strong reason not to award this particular badge in this scenario.

Things are not so true in the circular system, as previously explained. When you vote and predict, the best strategy is making your prediction and your vote the same. Even if we allowed people to make separate predictions from their votes, we would essentially be encouraging them to vote publicly – and one of the design goals is to encourage the opposite: surfacing unvoiced dissent. So, in circular teams, we must also have anonymous predictions. This makes that badge no different than "First vote!".

- "Fortune teller" (first prediction scored as "successful"), "Oracle" (scored n successful predictions), etc., or otherwise getting points for successful predictions.

  In the hierarchical model of teams, we clearly want to have managers that predict well, keep up on what's going on and enable their team to work productively. However, we do not want to encourage that so much that managers get in a competition with one another. A manager could simply "cheat" by be peeking over people's shoulders as the time to voting comes up. Of course, this puts unwanted pressure on people who will face the same pressures in their anonymous vote as they would if they spoke out in the open – defeating the purpose of the exercise. This is why it is safer not to tell the manager if and when he makes "successful" predictions: after all, exceptional skill in prediction does not necessarily map to successfully managing a team.

  In the circular model of teams, such badges would essentially be disastrous as they encourage and reward "everything is terrible"-style voting.

- "Condition green" (supervisor gets an alarm for the first time)

  In the case of highly hierarchical structures, a supervisor does not have such a role for a large number of teams. If such a badge was "awarded", voters would notice and start speculating on who's voting what, creating interest and gossiping that could very well work against the successful resolution of a situation. Even more critically, we think it's critically important to allow supervisors to receive an alert, study the situation and decide that it was probably a false alarm, a voting anomaly or that otherwise there is not enough evidence of unsurfaced unrest to unnecessarily unnerve an unsuspecting manager.

As a consequence attempts to gamify GEEZMO at best do not help and at worst directly work against its design tenets and goals. More research should be invested to clarify this research questions since gamification techniques have seen successful usage in the past, e.g., to assess software development progress or increase software development awareness [25].

6. RELATED WORK

Polling is now a very mature field of science, and much ink has been spilled on the subject.

Scholar research on repeated administration of the same poll is somewhat more sparse, and generally focused on correctly aggregating responses collected at separate points in time in order to *remove* the effects of time [7, 1, 19].

One attempt in exploiting the time dimension is instead proposed in the Delphi technique [3] that, for example, tries to administer the same comprehensive poll over and over with a basic response summary being provided to participants between each round.

The goal is to reach agreement between experts on a given topic in a pressure-free environment, as responses are anonymous. It appears, however, that the feedback loop provided by the response summary might be strong enough to cause GEEZMO to be self-defeating by creating pressure towards conformity [20].

Our research solution however is more closely related to the field of sentiment analysis or opinion mining. Indeed, much of said work stems from the MSR community [8]. For example, meta-algorithms that analyze sentiment from multiple bodies of text and classify them on a scale from 1 to 4 have been shown to work [17]. Work has also been done to classify shorter text snippets such as tweets according to a three-way polarity scale (positive/neutral/negative) [12].

However, we could not find any research about automated sentiment analysis of video or voice feeds (which could, conceivably, be based on simpler metrics than natural language processing uses); as a result, they would not be able to process all team communication (never mind the privacy consequences such a process would entail).

7. CONCLUSIONS

This paper elaborates on a High-Frequency Mood-Polling for Software Development "Weather Prediction" across software projects. Our research assumption is that using non-invasive mood-polling may be helpful in predicting sub-optimal organisational or social patterns taking place across a software development community. This paper also outlines our proof-of-concept prototype and how it was validated using simulations and statistical analysis methods.

Our preliminary validation shows interesting results into a tentative strategy at supporting team awareness on social and organisational issues across the organisational structure. I the future we plan to refine our prototype and deploy it in a practical experiment to evaluate its actual potential.

Also, we plan to further explore the limitations we currently evaluated for GEEZMO in terms of its fitness for usage within the scope of Gamification campaigns. We have observed there are severa limitations that might be overcome by investing additional research, e.g., investigating GEEZMO typical usage patterns and understand how they might be used as a basis for gamification.

8. ACKNOWLEDGEMENTS

The authors' work is partially supported by the European Commission grant no. 610531 (FP7 ICT Call 10), SeaClouds.

9. REFERENCES

[1] B. Blight and A. Scott. A stochastic model for repeated surveys. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 61–66, 1973.

[2] L. Bock. Passion, not perks. Thinking with Google, September 2011.

[3] N. C. Dalkey. Delphi. 1967.

[4] I. Epifani. Verifica di ipotesi. Appunti delle lezioni del corso di Statistica (2L) per gli allievi INF e TEL, AA 2008/2009, June 2009.

[5] S. Exchange. Company page: Stack exchange. Careers Stack Overflow, December 2014.

[6] S. Exchange. Stack exchange open source projects. Stack Exchange Blog, December 2014.

[7] M. Feder. Time series analysis of repeated surveys: The state–space approach. *Statistica Neerlandica*, 55(2):182–199, 2001.

[8] E. Guzman, D. Azócar, and Y. Li. Sentiment analysis of commit comments in github: An empirical study. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 352–355, New York, NY, USA, 2014. ACM.

[9] W. Heisenberg. Über den anschaulichen inhalt der quantentheoretischen kinematik und mechanik. *Zeitschrift für Physik*, 43(3-4):172–198, 1927.

[10] S. Invernizzi and S. Gatti. Adding awareness to a software forge: development of the titans approach and lessons learnt in participating in an apache open source development process. 2013.

[11] V. A. Kotelnikov. On the carrying capacity of the ether and wire in telecommunications. In *Material for the First All-Union Conference on Questions of Communication, Izd. Red. Upr. Svyazi RKKA, Moscow*, 1933.

[12] E. Kouloumpis, T. Wilson, and J. Moore. Twitter sentiment analysis: The good the bad and the omg! *ICWSM*, 11:538–541, 2011.

[13] W. A. LLC. Wolfram|alpha, November 2014.

[14] L. E. Lyberg, P. Biemer, M. Collins, E. D. De Leeuw, C. Dippo, N. Schwarz, and D. Trewin. *Survey measurement and process quality*, volume 999. John Wiley & Sons, 2012.

[15] J. A. Nelder and R. Baker. *Generalized linear models*. Wiley Online Library, 1972.

[16] J. Neyman and E. S. Pearson. *On the problem of the most efficient tests of statistical hypotheses*. Springer, 1992.

[17] B. Pang and L. Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of ACL*, pages 115–124, 2005.

[18] B. Pang and L. Lee. Opinion mining and sentiment analysis. *Foundations and trends in information retrieval*, 2(1-2):1–135, 2008.

[19] D. Pfeffermann. Estimation and seasonal adjustment of population means using data from repeated surveys. *Journal of Business & Economic Statistics*, 9(2):163–175, 1991.

[20] G. Rowe and G. Wright. The delphi technique as a forecasting tool: issues and analysis. *International journal of forecasting*, 15(4):353–375, 1999.

[21] D. Tamburri, P. Kruchten, P. Lago, and H. van Vliet. What is social debt in software engineering? In *Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on*, pages 93–96, May 2013.

[22] D. A. Tamburri, E. di Nitto, P. Lago, and H. van Vliet. On the nature of the GSE organizational social structure: an empirical study. *7th International Conference on Global Software Engineering*, pages 114–123, 2012.

[23] D. A. Tamburri, P. Lago, and H. van Vliet. Uncovering latent social communities in software development. *IEEE Software - Special Issue on Bridging Software Communities through Social Networking*, pages 26–32, 2012.

[24] D. A. Tamburri, P. Lago, and H. v. Vliet. Organizational social structures for software engineering. *ACM Computing Surveys (CSUR)*, 46(1):3, 2013.

[25] B. Vasilescu. Human aspects, gamification, and social media in collaborative software engineering. In P. Jalote, L. C. Briand, and A. van der Hoek, editors, *ICSE Companion*, pages 646–649. ACM, 2014.

The Role of Social Interactions in Value Creation in Agile Software Development Processes

Hiva Alahyari
Chalmers University of Technology
41296,Gothenburg
Sweden
hiva.alahyari@chalmers.se

ABSTRACT

This position paper presents an emerging research based on a set of expressed statements and impressions from conducted empirical research during the past few years. Agile software development emphasizes on social aspects through its methods and principles. In order to improve the processes within the organization and amongst various stakeholders, there is a need for social processes and various types of interactions to be studied in the context of agile development. The objective of this paper is to present the need to conduct more empirical studies to investigate the socialness of software engineering processes and in particular the role of various type of social interactions in improving development processes and therefore creating more value in organizations.

Categories and Subject Descriptors

D2 [Software Engineering], K.6.3 [Management of Computing and Information Systems]: Software Management: *Software Process*, K.6.1 [Management of Computing and Information Systems]: Project and People Management

General Terms

Human Factors, Management

Keywords

Agile software development, social processes, social interactions

1. INTRODUCTION

In Agile principles, the primary focus is on value creation. In agile development, what ever that doesn't add value, is considered waste [1].

SSE'15, September 1, 2015, Bergamo, Italy
© 2015 ACM. 978-1-4503-3818-9/15/09...$15.00
http://dx.doi.org/10.1145/2804381.2804384

Additionally, close collaboration and communication is essential to a successful agile team. Instead of 'big design up front', agile development encourages close collaboration with customer, iterative development and feedback. Agile software development highlights the social aspects of software development with explicit values, "such as communication and respect, and explicit principles, such as humanity and reflection" and social factors have shown to be an important factor in agile development [2,3]. Humans are social being and social interaction is the dynamic interplay of people within social structures. The importance of the role of social interactions have been taken into consideration in various fields of study such as economy and a large body of research has considered the role of social interactions in this field [4]. In software engineering world also, there are filed of study such as Global Software engineering [5] and Open source development [6] that have addressed various factors such as the role of cultural differences or the sense of community and their influence on the development processes and the expected outcomes**.** A study by Whitworth and Biddle, suggests that there are strong social forces at play in agile teams that underline the value of agile methodologies [3]. In addition to [3] which has emphasized the role of social processes in agile development, several studies have reported human aspects and people factor, the role of team works and team dynamics and also other aspects such as individual motivations, communication and knowledge sharing, and even social networks in agile software development processes [7,8,9,10,11].

All the previously mentioned studies [3,4,5,6,7,8,9,10,11,12] in agile software development or other fields such as global software engineering or open source development have emphasized the role of social processes, have studied social aspects, or observed one or more types of social interactions and have stated the need for further research on the role of the studied social aspects. However the studied social processes are not associated with the classification of social interactions existing in sociology for example. To the best of our knowledge, in agile software development processes no studies have been dedicated to study the various types of social interactions (based on the existing classifications and terminologies) amongst various players and with a focus on value creation. This paper presents a need for further future research that considers social processes and various types of social interactions amongst the members and different stakeholders and as a part of software development processes. Based on this need and an existing classification of social interactions, a set of research questions are suggested at the end of the paper.

The reminder of this paper is organized as follow. Section 2,is an introduction to social process and the five known types of interactions. Section 3 presents the inspirations and how and why we think that there is a need for the research topic. And finally section 4 suggests a set of research questions.

2. SOCIAL PROCESSES AND TYPES OF INTERACTIONS

A social process refers to the type of existing relationship between individuals and groups. Sociologists have suggested different classification of social processes. One of the most common kinds is to classify it into five types of Cooperation, Conflict, Competition, Accommodation and Assimilation [13].

2.1 Cooperation

3. Cooperation is one of the fundamental processes of social life and it is an integrating activity. It means working together in the pursuit of common interests or common goal. According to some functionalists, cooperation for human beings is both a psychological and a social necessity [13].

4. There are two modes of cooperation in a social life, which are direct/primary and indirect/secondary. It is primary when cooperation is both a means and an end as it is a highly valued goal like in a family.

5. It is called secondary when cooperation by itself is not a value but a means to an end like in a business when people perform tasks towards a single end [13].

2.2 Conflict

Contrary to the cooperation, conflict is a struggle over values, power and scarce resources. The aim of conflicting persons or groups is not only to gain the desired values but also to win over the rivals or eliminate them. Conflict is universal and occurs in all places and at all time.

Conflict generally occurs due to individual differences, cultural differences, clash of interests and social changes. Conflict might also serve as constructive and positive ends according to some of the sociologists (G. Simmel and L. Coser) [13].

2.3 Competition

Competition is fundamental forms of struggle in strive against each other for the possession of or use of some limited material and non-material good.

As long as attention of competitors is confined to goals, it remains as an impersonal manner.

Most sociologists view competition as a positive thing that can motivate people to achieve goals. Competition tends to stimulate economy, efficiency, and inventiveness. It also tends to increase one's ego and to give one satisfaction.

However, competition can also lead to psychological stress, a lack of cooperation in social relationships, inequality and even conflicts [13].

2.4 Accommodation

Accommodation means resolution of conflict by adjusting oneself to the new environment. Accommodation is the term used by the sociologists to describe the process by those once in conflict who can work together in common enterprises. It brings arrangements, which permit groups to work together [13].

2.5 Assimilation

Assimilation on the other hand, is a process of interpenetration of cultural life. It is a process whereby persons and groups acquire the culture of the other persons and groups become similar and identified in their interests and values.

But at the same time, the extreme differences in cultural background, prejudice and physical differences usually act as barriers to assimilation [13].

3. THE ROLE OF SOCIAL INTERACTIONS IN VALUE CREATION

During the last three years, we have conducted more than 75 interviews, in various case companies and with different industry practitioners, in order to investigate the software development processes either from the development teams perspective or the organizational perspective. The case companies and in particular the interview subjects were following agile methods and the companies were in the process of scaling agile to all the teams and different parts of the organization. Some of these case studies have been published [14,15,16] and the most recent one is in the submission process [17]. The interviewees were involved in software development processes in one way or other and their role varied. Their role could be a project manager, product owner, software developer, or software architect/designer/tester.

Various aspects have been investigated during these interviews. In some interviews, study participants have been asked to talk about barriers in moving towards agile way of working, continuous integration and continuous deployment. Amongst the answers, mindset of people was pointed out as a barrier. People did not have a very collaborative mindset and this made it difficult. Some barriers were regarding the information and some had raised the need for a better process in regards to receiving the needed information. These aspects could be considered under the **cooperation** category.

Some said that they receive the needed information through social interactions and through informal situations such as "by the coffee machine" [14]. In another study which is about customer specific teams at a large telecommunication company, some of the interviewees believed that the fact that their customers have access to a team, that they can directly talk to and interact with, increases their customers' satisfaction level independent from the quality of the developed feature for example [15]. In those examples sounds that **social interactions in general** play as a factor.

During the interviews for the recent study [17], which investigates the concept of value in 14 development organizations, some of the interviewees did not find the **cooperation** within distributed development processes very satisfactory. Working with suppliers was not always easy and they considered it as hinder towards some values such as fast feedback and deliveries. However, distributed and borderless development teams are not a new concept and are a part of development processes in many big organizations, nowadays.

Particularly in regards to the suppliers, differences in culture were mentioned as a time and energy consuming aspect. Depending on the influential factors, this problem could be seen from **accommodation** aspect or **assimilation aspect** or may be both.

On the other hand, some other teams mentioned the existing **cooperation** within their team members, as one of the strengths and value creation factors and were very happy about it.

Competition in gaining market shares and so delivering quality products in time is a known situation for software development organizations. In addition to the existing competition between various companies and to gain more revenue, we received the impression that concept of "Team" in agile development, which is working in cross-functional independent teams, might create another sort of **competition**. That is the competition of being the better and faster and the "more agile" team than the others.

Conflicts were also another aspect, which were expressed in between the lines. Though, it is hard to know if it is a conflict or just a perception of conflict. Team members were very focused on what is value for their own team and their team performance. While the management decisions did not seem very inline with those internal/team values, sometimes.

4. WHAT ADDS VALUE AND WHAT CREATES WASTE AND UNDER WHAT CIRCUMSTANCES

Based on the expressed statements and impressions received during the conducted interview studies, we believe that there is an emerging need to further study social processes and the type of social interactions in software development organizations and in particular agile. In the examples from the interviews mentioned in section 3, we can see that various types of social interactions has been stated and or played a role in the social interactions amongst various players.

Therefore we think that there is a need to firstly study what kind of social processes exists in agile way of working or what type of social interactions appears more commonly.

Secondly, to investigate in depth what are the impacts of different types of social interactions and to what extent they influence the software development processes and the dynamic between various agile teams.

Almost all the five types of social interactions mentioned in section 2, could have the both positive and negative sides based on their context.

Recalling that creating value and reducing waste is the primary goal for an agile organization, there is a need to further investigate how and when each of the various types of interactions can act as a source of value and when they can be considered as waste.

For example competition could be both constructive /value creating or destructive depending on how and where it occurs.

If we consider cooperation as a positive only aspect, we still need to know if it is equally important everywhere. Where cooperation is the most important? Is it between different stakeholders, between the teams and organization or between the teams and customer? And which one can create more value?

Is the conflict (of interests amongst various stakeholders) always a negative factor or it can be also used as triggers towards more solution thinking or creativity for example?

Increasing our knowledge on this regard, not only assists us to address various improvement needs in the processes, but also might elevate a new condition. It could raise a need for reconsideration of some of the methods and ways of handling agile teams or the processes amongst the organization and various stakeholders.

Therefore, we think that there is a need to conduct more empirical studies within software development organizations and in particular agile organizations to investigate the various aspects of social software engineering.

We suggest a set of research questions as follow:

- Which of the five types of the social processes/interactions exist or occurs more commonly in agile development processes?
- Which of the five types of social processes exits amongst various stakeholders and what types of interactions occur more commonly amongst them?
- Which types of interactions are the most important in improving the agile software processes?
- How the effects of various social interactions can be used to improve software engineering processes?
- What are the tradeoffs between the five types of social interactions in terms of value creation in agile software development processes?
- Which of the interaction types satisfy various stakeholders and in particular the customers the most?
- How social interactions can be assessed or evaluated as parts of software engineering processes?
- How can we define some metrics for measuring the effect of various types of social interactions?
- Which agile practices promote one or more types of social interactions and how?
- Can agile practices be classified into various categories based on their effect and potential in relation to the five types of social interactions?
- Could or should various agile practices be recommended or eliminated based on their potential in promoting various types of social interactions?

5. REFERENCES

[1] M. Poppendieck and T. Poppendieck, *Lean Software Development: An Agile Toolkit*. Boston, Mass.: Addison-Wesley Professional, 2003

[2] Beck, K., Andres, C., 2004. Extreme Programming Explained: Embrace Change. Addison-Wesley, San Francisco

[3] E. Whitworth and R. Biddle, "The Social Nature of Agile Teams," in *Agile Conference (AGILE), 2007*, 2007, pp. 26–36.

[4] W. A. Brock and S. N. Durlauf, "Discrete Choice with Social Interactions," *The Review of Economic Studies*, vol. 68, no. 2, pp. 235–260, Apr. 2001.

[5] J. Kotlarsky and I. Oshri, "Social ties, knowledge

[6] sharing and successful collaboration in globally distributed system development projects," European Journal of Information Systems, vol. 14, no. 1, pp. 37–48, 2005.

[7] K. Crowston and J. Howison, "The social structure of free and open source software development," First Monday, vol. 10, no. 2, Feb. 2005.

[8] T. Chau and F. Maurer, "Knowledge Sharing in Agile Software Teams," in *Logic versus Approximation*, W. Lenski, Ed. Springer Berlin Heidelberg, 2004, pp. 173–183.

[9] A. Martin, J. Noble, and R. Biddle, "Experience on the Human Side of Agile," in *Agile Processes in Software Engineering and Extreme Programming*, vol. 9, P. Abrahamsson, R. Baskerville, K. Conboy, B. Fitzgerald, L. Morgan, and X. Wang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 234–235.

[10] A. Cockburn and J. Highsmith, "Agile software development, the people factor," *Computer*, vol. 34, no. 11, pp. 131–133, Nov. 2001.

[11] G. Asproni, "Motivation, teamwork, and agile development," *Agile Times*, vol. 4, no. 1, pp. 8–15, 2004.

[12] T. Dingsoyr and T. Dybå, "Team effectiveness in software development: Human and cooperative aspects in team effectiveness models and priorities for future studies," in *2012 5th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2012, pp. 27 –29.

[13] W. Picard, "Social Protocols for Agile Virtual Teams," in *Leveraging Knowledge for Innovation in Collaborative Networks*, L. M. Camarinha-Matos, I. Paraskakis, and H. Afsarmanesh, Eds. Springer Berlin Heidelberg, 2009, pp. 168–176.

[14] Subberwal, Dictionary Of Sociology. Tata McGraw-Hill Education.

[15] H. H. Olsson, H. Alahyari, and J. Bosch, "Climbing the 'Stairway to Heaven ' – A Multiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software," in *2012 38th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, 2012, pp. 392–399.

[16] H. H. Olsson, J. Bosch, and H. Alahyari, "Customer-Specific Teams for Agile Evolution of Large-Scale Embedded Systems," in *2013 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, 2013, pp. 82–89.

[17] H. H. Olsson, J. Bosch, and H. Alahyari, "Towards R& D as Innovation Experiment Systems: A Framework for Moving Beyond Agile Software Development," Proceedings of the IASTED International Conference on Software Engineering, SE 2013 pp. 798-805. 2013.

[18] H.Alahyari, R.B.Svensson and T.Gorschek, "A study of Value in Agile Software Development Organizations" (In submission) *submitted to Transactions in Software Engineering,* June 2015.

Could Social Factors Influence the Effort Software Estimation?

Valentina Lenarduzzi
University of Bolzano-Bozen
Piazza Domenicani, 3
39100 Bolzano/Bozen - Italy
valentina.lenarduzzi@unibz.it

ABSTRACT
Effort estimation is often influenced by several factors, including social. This study aims at understanding the interactions between social factors and effort during effort estimation. I want to analyze the dynamics that occur when a developer estimates the effort for a specific task and the influence of the work team and the work conditions.

I conducted a semi-structured interview among three different projects with different developers working in Agile and Scrum processes, asking them which factors and social aspects they take in to account when they estimate the effort during the development processes.

Results show an important influence of social factors during the effort estimation phase, and call for future works for a large scale Survey for a more accurate identification.

Categories and Subject Descriptors
D.2.9 [**Management**]: *Cost estimation, Life cycle*

General Terms
Effort Estimation, Measurement, Experimentation, Human Factors.

Keywords
Effort estimation, Agile process, Scrum process, social factors.

1. INTRODUCTION
Effort estimation is one of the most important activities in any domain, especially in software engineering where still a big number of projects fails because of effort estimation errors[1].

Several effort estimation models have been defined based on user experience or on historical data. In both cases the evaluation is based on factors that are difficult to identify. Those factors are usually related to the project to be developed, the context, the knowledge of domain and last, but not least, social factors such as the interaction in the development team.

In my work, I want to investigate which are the social factors that influence the effort estimation process. I analyze the personal mechanism that occurs when a developer estimates the effort for a specific task and the influence of the work team and the work conditions.

I conduct a semi-structured interview for collecting which factors the developers take in to account during the effort estimation and I analyze three different projects with different team composed by experts and students.

Results show that most of the interviewed developers consider communication and work pressure as very important social factors while other factors such as familiarity with the project and managerial skills are not considered as relevant social factors during the effort estimation phase.

The remainder of this paper is organized as follows: Section 2 describes related work. Section 3 describes the study carried out. Section 4 shows the results. Section 5 concludes.

SSE'15, September 1, 2015, Bergamo, Italy
© 2015 ACM. 978-1-4503-3818-9/15/09...$15.00
http://dx.doi.org/10.1145/2804381.2804385

2. RELATED WORKS
Software effort estimation is a complex and critical [2] task. Generally the project's effort is estimated based on the effort of projects developed in the past. A big set of information should be also taken into account to estimate the effort, such as the project size, the domain, and many other factors such as law constrains, standard and others that may significantly influence the estimation. The parameters identification and measurement is very complex, since software products are less tangible than the classical engineering ones and the requirement volatility increasing the complexity of the effort estimation negatively influencing the estimation accuracy. [3]

Several estimation models have been defined in the past, mainly based on developers' experience [4] or statistical techniques on historical data [5, 6, 7].

Expert-based estimation models are based on people's experience where the estimation is commonly carried out by analogy, comparing the project to be developed to similar past projects [3] Data-driven models are based on statistical or machine-learning techniques with the goal of reducing the subjectivity of the expert evaluation and automating the effort estimation as much as possible. [5, 6, 7].

Here I report on existing work that analyze the information taken into account by developers during the effort estimation process.

Molokken [8] conducted a systematic literature review of effort estimation identifing several cost factors related to the subjective such as work pressure and communication process. Lamersdorf et al [12] identified as social factor language and cultural differences, communication process and competence level while Kroll et al [13] identified the communication, the team structure

and the work pressure. Also Da Silva et al [14] recognized language and cultural differences but considered in addition the work dispersion and the team size. Finally Nurdiani et al [15] highlighted language and cultural differences and communication issues.

Another set of factors has been identified by Popli and Chauhan [10]. They call "People-Related" factors working time and experience of previous project.

In order to improve the estimation accuracy Menzies et al. [9] and Jørgensen [11] defined some best practices for expert based effort estimation models particularly for personnel.

Another studies, carried out by Taibi et Al, also confirm language and cultural differences, communication and working time as main social influencing factors during the effort estimation [16, 17, 18, 19].

In Table 1, I summarize the most important factors identified here.

Table 1. social effort estimation factors

Factors	Study id.
Language and Cultural Differences	[8] [12] [14] [15] [16] [17] [18]
Communication	[8] [10] [13] [15] [16] [17] [18]
Communication Process	[8] [12] [16][17][18]
Team structure	[8] [13]
Work Pressure	[8] [9] [11] [13]
Work Dispersion	[8] [14]
Team Size	[8] [14]
Competence Level	[8] [12]
Familiarity in Team	[10]
Managerial Skill	[10]
Working Time	[10] [16] [17] [18] [19]
Experience of Previous Project	[10]
Technical ability	[9] [10] [11]

3. THE INTERVIEW

In order to understand if any social factors could influence the effort estimation I conducted three set of semi-structured interviews, among the developers of three different projects developed with Agile methodologies. In this section, I present the goal and the research question. Then I describe the study design, the study preparation and the procedure.

3.1 Study goal and Research Question

Accordingly to our expectation I formulated the goal as following:

Analyze social factors
for the purpose of evaluating the influence of Social factors
from the viewpoint of developers
in the context of effort estimation

This leads to our research question:
RQ: which social factor influence the effort estimation?

3.2 Study design

The study was designed as semi-structured interviews with students and researcher as participants and I carried it out by means of a Questionnaire. The Questionnaire is defined in collaboration with experts in empirical software engineering from the University of Kaiserlautern and University of Bolzano/Bozen.

The questionnaire is composed by three sections. In the first section I profile the interviewees, so as to gather information on their profile. In the second section I ask to rate each factors listed in Table 1 with a likert scale ranging from 1 to 5, where 1 means not important and 5 very important. In the third section, I ask to list any additional factor that could influence the effort estimation.

The study has been designed to be replicated several time, with the same settings, in different projects.

3.2.1 Study preparation

All participants were master students and expert developers. In the first round of interviews the team was a distributed teams in several projects with part-time developers working during non-overlapping hours. In addition, at the University of Kaiserslautern, the development is often gave to students with part-time contracts and they work for a small number of hours per week, in their spare time.

In the second and third set of interviews, the team was always students but they worked for a minimum of six hours per week and mostly during the lectures.

3.2.2 Study Execution

Here I report on the three projects considered in the study.

The first project (GROUP 1) was developed in the context of a web application developed by six developers at the University of Kaiserslautern (Germany) [20]. The project has been developed following the Moonlight Scrum [16, 17, 18, 19] process, by Master Students. The experience level was the same for the team, even if nobody had any experience with agile methodologies. They worked part-time, with weekly working hours between four and ten, with a total effort spent of 39 hours per week. The development started in February 2013 until the end of May 2013.

The second (GROUP2) and the third (GROUP 3) project were conducted in the context of the development of two Master projects, during the course of "Software Factory" in the Computer Science department of the Free University of Bolzano/Bozen (Italy). Both groups developed a web application but, while one group developed in Java /JSP the second developed the application in Asp.Net/C#. Both groups are composed by three developers, working for 10 hours a week, from March 2015 until the end of May 2015.

4. RESULTS ANALYSIS

Here I report the results for all groups and the aggregated results.

Analyzing the results presented in Table 2, users considered very important communication (4.25 out of 5) and work dispersion (4.42 out of 5) while eight factors are considered moderately important or of little importance with likert-scale values that range from 3 to 4 (Domain Knowledge, Communication Process, Experience of Previous Project, Working Time, Language and Cultural Differences, Work Dispersion, and Work Pressure).

Managerial skills, familiarities with the team and competence level are only considered of little importance while only the familiarity with the project is considered unimportant.

Table 2. Results analysis

Factors	ALL GROUPS			GROUP1			GROUP2			GROUP3		
	Mean	*Median*	*Std.dev*	*Mean*	*Median*	*Std.dev*	*Mean*	*Median*	*Std.dev*	*Mean*	*Median*	*Std.dev*
Work Pressure	4,42	4	0,51	4,67	4	0,52	4,67	4	0,52	4,42	4	0,51
Communication	4,25	4	0,62	4,33	4	0,75	4,33	4	0,75	4,25	4	0,62
Work Dispersion	3,58	4	1	3	4	0,75	3	4	0,75	3,58	4	1
Language and Cultural Differences	3,5	3	1,24	3,33	4,5	1,26	3,33	4,5	1,26	3,5	3	1,24
Communication Process	3,42	4	1,31	2,67	4	0,75	2,67	4	0,75	3,42	4	1,31
Experience of Previous Project	3,42	3,5	1,16	2,67	4	0,75	2,67	4	0,75	3,42	3,5	1,16
Working Time	3,42	3,5	1,08	3,33	4	0,63	3,33	4	0,63	3,42	3,5	1,08
Domain Knowledge	3,33	3,5	1,07	2,67	4	0,75	2,67	4	0,75	3,33	3,5	1,07
Team Structure	3,33	3	1,07	3,67	3	1,17	3,67	3	1,17	3,33	3	1,07
Technical ability	3,17	3	0,72	3	3	0,63	3	3	0,63	3,17	3	0,72
Competence Level	2,92	3	0,79	3,67	2,5	0,55	3,67	2,5	0,55	2,92	3	0,79
Familiarity in Team	2,83	3	1,11	3,67	2	0,75	3,67	2	0,75	2,83	3	1,11
Managerial Skill	2,08	2	0,9	1,33	3	0,52	1,33	3	0,52	2,08	2	0,9
Familiarity with the project	1,83	2	0,72	1,33	2	0,63	1,33	2	0,63	1,83	2	0,72
5-point ordinal Likert scale: 1=not important, 2=Of little importance, 3=moderately important, 4=important, 5=very important												

5. CONCLUSION AND FUTURE WORK

In this paper, I highlight the problem of social factors during the effort estimation phase, providing first a literature review to understand the social factors commonly considered important during the effort estimation phases. There are several works reporting on different models to estimate the effort but, at the best of our knowledge, no studies analyzed the influence of social factors.

For this reason, I designed and executed a semi-structured interview, I carried out on three projects, asking our developers which social factor they consider during the effort estimation phase.

Results show that most of the interviewed developers consider Communication and work pressure as very important social factors while other factors such as familiarity with the project and managerial skills are not considered as relevant social factors during the effort estimation phase.

As for threats to validity, results are based only on the analysis of three development processes, based on a relatively short timeframe (3-4 months). Developers are master students, with a limited experience (2-3 years) in software development.

Future work include the execution of a large scale survey on senior developers, so as to target a broader population of software engineers and to better understand which factor influence more the effort estimation with the ultimate goal of increasing the estimation accuracy.

Moreover, I am planning to setup a project to understand to which extent social factors influence software effort estimation, so as to better take them into account during the estimation process.

6. REFERENCES

[1] Lenarduzzi, V. Morasca, S. Taibi, D. "Estimating Software Development Effort Based on Phases", In 39th Euromicro Conference on Software Engineering and Advanced Applications, 2014.

[2] Saleem, B. and Dhavachelvan, P. 2010. Analysis of Empirical Software Effort Estimation Models.International Journal of Computer Science and Information Security. 7, 3, pp.68-77.

[3] Khatibi, V and Jawawi, D. 2010. Software Cost Estimation Methods: A Review. Journal of Emerging Trends in Computing and Information Sciences. 2, 1, 21-29.

[4] Molokken, K and Jorgensen M. 2003. A review of surveys on software estimation. ISESE - International Symposium on Empirical Software Engineering. 223-230.

[5] Jeffrey, R., Ruhe, M. and Wieczorek, I. 2001. Using Public Domain Metrics to Estimate Software Development Effort. IEEE International Software Metrics Symposium. 16-27.

[6] Jorgensens, M. and Sjoberg, D.I.K. 2003. An effort prediction interval approach based on the empirical

distribution of previous estimation accuracy. Information and software Technology. 45,123-136

[7] Sentas, P., Agelis, L., Stamelos, I. and Bleris, G. 2005. Software productivity and effortprediction with ordinal regression. Information and software technology. 47, 17-29

[8] Molokken, K. 2004. A review of studies on expert estimation of software development effort. The journal of system and software. 70, 37-60.

[9] Menzies, T., Chen, Z., Hihn, J. and Lum, K. 2006. Selecting Best Practices for Effort Estimation. IEEE Transactions on Software Engineering. 32(11), 883-895.

[10] Popli, R. and Chauhan, N. 2013. Agile Estimation Using People and Project Related Factors. International Conference on Computing for Sustainable Global Development (INDIACom). 564-569.

[11] Jorgensens, M. 2005. Practical Guidelines for Expert-Judgment-Based Software Effort Estimation. Software IEEE. 22(3), 57-63.

[12] Lamersdorf, A., Munch, J., Torre, A.F.V., Sanchez, C.R. and Rombach, D. 2010. Estimating the Effort Overhead in Global Software Development. In Proceedings of 5th IEEE International Conference on Global Software Engineering - ICGSE'10, Princeton, USA. 267–276.

[13] Kroll, K., Audy, J.L.N. and Prikladnicki, R. 2011. Mapping the Evolution of Research on Global Software Engineering - A Systematic Literature Review. In International Conference on Enterprise Information Systems - ICEIS 2011, Beijing, China. 260–265.

[14] Da Silva, F.Q.B., Costa, C., Franca, A.C. and Prikladinicki, R. 2010. Challenges and solutions in Distributed Software Development Project Management: A systematic literature review. In Proceedings of 5th International Conference on Global Software Engineering - ICGSE 10, Princeton, USA. 87–96.

[15] Nurdiani, I., Jabangwe, R., Smite, D. and Damian, D. 2011. Risk Identification and Risk Mitigation Instruments for Global Software Development: Systematic Review and Survey Results. In Proceedings of 6th IEEE International Conference onGlobal Software Engineering Workshop - ICGSEW'11, Helsinki, Finaland. 36–41.

[16] Diebold P., Lampasona C. and Taibi D. 2013. Moonlight Scrum: An Agile Method for Distributed Teams with Part-Time Developers Working during Non-overlapping Hours" ICSEA'11 The Eighth International Conference on Software Engineering Advances. 318-323.

[17] Lenarduzzi, V. and Taibi, D. 2014. Can Functional Size Measures Improve Effort Estimation in SCRUM? ICSEA - International Conference on Software Engineering and Advances.

[18] Lenarduzzi, V., Lunesu, I., Matta, M. and Taibi, D. 2015. Functional Size Measures and Effort Estimation in Agile Development: A Replicated Study. International Conference on Agile Software Development. 212, 105-116.

[19] L. Lavazza, S. Morasca, D, Taibi, and D. Tosi, "Applying SCRUM in an OSS Development Process: An Empirical Evaluation", in 11th International Conference, XP 2010, 2010, pp. 147-159

[20] D. Taibi, Lenarduzzi, V. , Dieudonné, L. , and Plociennik, C. "Towards a Classification Schema for Development Technologies: an Empirical Study in the Avionic Domain", International Journal On Advances in Software, vol. 8, no. 1&2, 2015.

Understanding the Affect of Developers: Theoretical Background and Guidelines for Psychoempirical Software Engineering

Daniel Graziotin
Faculty of Computer Science,
Free University of
Bozen-Bolzano
Piazza Domenicani 3
Bolzano/Bozen, Italy
daniel.graziotin@unibz.it

Xiaofeng Wang
Faculty of Computer Science,
Free University of
Bozen-Bolzano
Piazza Domenicani 3
Bolzano/Bozen, Italy
xiaofeng.wang@unibz.it

Pekka Abrahamsson
Department of computer and
information science
Norwegian University of
Science and Technology
NO-7491 Trondheim, Norway
pekkaa@ntnu.no

ABSTRACT

Affects—emotions and moods—have an impact on cognitive processing activities and the working performance of individuals. It has been established that software development tasks are undertaken through cognitive processing activities. Therefore, we have proposed to employ psychology theory and measurements in software engineering (SE) research. We have called it "psychoempirical software engineering". However, we found out that existing SE research has often fallen into misconceptions about the affect of developers, lacking in background theory and how to successfully employ psychological measurements in studies. The contribution of this paper is threefold. (1) It highlights the challenges to conduct proper affect-related studies with psychology; (2) it provides a comprehensive literature review in affect theory; and (3) it proposes guidelines for conducting psychoempirical software engineering.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*Productivity, Programming Teams*; H.1.2 [**Models and Principles**]: User/Machine Systems—*Human factors, Software psychology*; J.4 [**Social and behavioral Science**]: [Psychology]

Keywords

Affects, emotions, moods, human aspects in software development, psychology of programming, psychoempirical software engineering

1. INTRODUCTION

The Nobel prize winner Daniel Kahneman has pointed out that it is unrealistic to limit our understanding of human behaviors solely through rational models [41]. Yet, software engineering (SE) research has been known to be too much confined in the fallacy of rationality-above-everything paradigm [63], to miss out the possibility to be a social discipline [82], and to focus too much on domains of technical nature while neglecting the so-called *soft aspects* or *human-related* topics [52]. But software development *is* a very human activity. Software development happens in our minds first, then on artifacts [26]. It has been established that development is intellectual, and it is carried out through cognitive processing activities [25, 26, 42]. Indeed, we are human beings, and, as such, we behave based on affect as we encounter the world through our emotions and moods [10]. The affects pervade organizations by coloring the workers' thoughts, and they influence their behavior [8]. Affects have a role in the relationships between workers, deadlines, work motivation, sense-making, and human-resource processes [3]. Although affects have been historically neglected in the studies of industrial and organizational psychology [61], an interest in the role of affects on job outcomes has accelerated over the past fifteen years in psychology research [27]. While research is still needed on the impact of affects on cognitive activities and work-related achievements in general, this link undeniably exists according to psychology research.

We have shown elsewhere [33] that practitioners are deeply interested in their affects while developing software, which causes them to engage in long and interesting discussions when reading related articles. Thus, it is important to understand the role of affects in software development processes. Even more, we share the view of Lenberg et al. [52] that SE should also be studied from a behavioral perspective. We have, in fact, focused on these issues for some time now, by producing several articles on this avenue, i.e., [30, 16, 31, 32, 33, 34]. We have also proposed the term *psychoempirical software engineering* [35] to denote research in SE with proper theory and measurement from psychology. Our message was well-received by the community with some degree of agreements regarding terminology, e.g., [51, 52]. However, we show below that long is the road to properly address the human aspects of SE with psychology.

Problem: SE Lacks in Theoretical Background of Affects and Guidelines for Using Psychology.

SSE'15, September 1, 2015, Bergamo, Italy
ACM. 978-1-4503-3818-9/15/09...$15.00
http://dx.doi.org/10.1145/2804381.2804386

Given the rising number of recent SE articles that deal with the affects of developers, e.g., [62, 28, 37, 20], we believe that it should be important for researchers to adopt a critical view of the phenomenon under study, and that they do not fall into the several misconceptions when dealing with the affect of developers [34].

Yet, we understand that we have placed ourselves in a "very confused and confusing field of study" ([64], p. 2). We experienced this confusion especially during our talks at ISERN 2014, where we chaired a workshop called psychoempirical SE [35], and during the CHASE 2015 workshop [6], where we presented some common misconceptions and measurements of the affect of software developers [34]. Such misconceptions include confusing affect and the related constructs of emotions and moods with motivation or job satisfaction, which has happened even in articles already dealing with misconceptions of motivation with respect to job satisfaction, e.g., [29], although affects were not the focus of the study in this case.

Other issues lie in missing out the opportunity of using validated measurement instruments for affect. An example is the use of the niko-niko calendar for assessing the mood of a software development team, e.g. [77], or the so-called happiness index, e.g., [56]. Another example of the missed opportunity is when a single truth is assumed in the writing of articles, like in a CACM positional article claiming that "psychologist recognize eight basic emotions, with each positive balanced by a negative", e.g. "love-hate" ([19], p. 34), or in a proper empirical study where it has been claimed that "there are six basic emotions or universal emotions: anger, happiness, fear, [..]" ([13], p. 1079). We will show below that it is not true that a unique dominant, accepted theory exists for affect, emotions, and moods. Researchers should recognize this issue when employing such delicate concepts for conducting research.

Proposal: Theoretical Background of Affects and Guidelines for Psychoempirical SE.

While it would be preposterously arrogant on our side to claim the all-encompassing knowledge of the topic, we would like to share what we have learned so far in our journey to understanding software developers through their affect. This article builds upon our experience, the feedback collected at our talks and peer review processes, and the previously conducted research, to build some theoretical background for understanding the affect of software developers. We draw from research in psychology in the last decades, and offer a comprehensive review of the theory of affect (section 2) and, as a follow-up of our ISERN 2014 workshop [35], we propose our guidelines for psychoempirical SE (Section 3) for conducting studies in SE with psychological theory and measurement.

2. AFFECT, EMOTIONS, MOODS: THEORETICAL BACKGROUND

The fields of psychology have failed to agree on the definitions of affects and the related terms such as emotions, moods, and feelings [64, 72]. Yet, it is desirable that we provide a starting set of definitions, which we will however criticize.

Let us start by stating that the term *affect* (or affective state) has been defined as "any type of emotional state [...] often used in situations where emotions dominate the person's awareness" [85]. This definition is problematic as it contains the term *emotion*, which has not yet been defined, and it does not help in defining the (now apparently) superconstruct *affects*. Indeed, the term *affects* is often associated in the literature with *emotions* and *moods*. We now are left with three terms, which look remarkably similar to each other.

Plutchik [66] has defined *emotions* as the states of mind that are raised by external stimuli and are directed toward the stimulus in the environment by which they are raised. However, Kleinginna et al. [45] reported one year later that more than 90 definitions have been produced for this term, and no consensus in the literature has been reached. The term has been taken for granted and often defined with references to a list, e.g. anger, fear, joy, surprise [9]. To worsen this, *emotion* as a term is not universally employed, as it is a word that does not exist in all languages and cultures [71].

Moods have been defined as emotional states in which the individual feels good or bad, and either likes or dislikes what is happening around him/her [65]. Yet again, a definition of one construct contains another construct of our interest.

How Do Emotions and Moods Differentiate, Then?

While for some researchers certain moods are emotions and vice versa [18], it has been suggested that a distinction is not necessary for studying cognitive responses that are not strictly connected to the origin of the mood or emotion [89]. Distinctions between emotions and moods are clouded, because both may feel very much the same from the perspective of an individual experiencing either [5] and are now a part of common sense [72]. They are embedded in psychologists' questions and, as a consequence, answers. Reisenzein [68] argued that "*the consensual definition of emotion is not a precondition but the result of scientific research; and even then, it remains a revisable empirical hypothesis*" (p. 2). So, affects, emotions, and moods are an emergent construction rather than a latent entity [11, 58].

We have adopted the same stance of several researchers in the various fields [79, 78, 88, 17] and employed the noun *affects* (affective states) as an umbrella term for emotions and moods. We will show that, according to a recent unifying theory, this strategy does make sense.

2.1 The Major Frameworks for Affect Theories

According to Huang [39], four major theories exist for emotions (moods, affects) in psychology. However, we see that these four theories and all the other we could review fall into two competing frameworks.

The Discrete Framework.

One framework, namely the discrete approach, collects a set of basic affective states that can be distinguished uniquely [66], and that possess high cross-cultural agreement when evaluated by people in literate and preliterate cultures [23].

The Differential Emotions Theory [40] states that the human motivation system is based on ten fundamental emotions (interest, joy, surprise, distress, anger, disgust, contempt, fear, shame, and guilt). These fundamental emotions function for the survival of human beings, possess an own neural network

in the brain, and an own behavioral response. Finally, these emotions interact with each other simultaneously.

Ekman [23] proposed a set of basic affects, which include anger, happiness, surprise, disgust, sadness, and fear. However, the list has received critique, leading to an extended version of eleven elements [24]. They include amusement, embarrassment, relief, and shame.

In the Circular Model of Emotion [66], a structure describing the interrelations among emotions has been proposed. Eight primary, bipolar affective states were presented as coupled pairs: joy versus sadness, anger versus fear, trust versus disgust, and surprise versus anticipation. These eight basic emotions vary in intensity and can be combined with each other, to form secondary emotions. For example, joy has been set as the midpoint between serenity and ecstasy, whereas sadness has been shown to be the midpoint between pensiveness and grief. Emotions can vary in intensity and persistence (to form moods, for example). Emotions, under this theory, serve an adaptive role in dealing with survival issues.

Developing a minimal list of basic affective states appears to be difficult with the discrete approach. Subsequent studies have come to the point where more than 100 basic emotions have been proposed [80].

The Dimensional Framework.

The dimensional framework groups affects in major dimensions that allow a clear distinction among them [70, 47]. In the PAD models, three dimensions of Pleasure-displeasure, Arousal-nonarousal, and Dominance-submissiveness [75, 70, 57] characterize the emotional states of humans. *Valence* (or pleasure) is the attractiveness (or adverseness) of an event, object, or situation [53] [49]. The term refers to the "direction of a behavioral activation associated toward (appetitive motivation) or away (aversive motivation) from a stimulus" [47]. *Arousal* represents the intensity of emotional activation [47]. It is the sensation of being mentally awake and reactive to stimuli, i.e. vigor and energy or fatigue and tiredness [90]. *Dominance* (or control, over-learning) represents a change in the sensation of the control of a situation [7]. It is the sensation by which an individual's skills are perceived to be higher than the challenge level for a task [15]. Figure 1 provides a representation of a PAD model of valence and arousal, and examples of related discrete affects with an indication to where they might correspond on the axes.

Emotional states under the PAD models include moods, feelings, and any other feeling-related concepts. The dimensions are usually bipolar, indicating that the presence of pleasure excludes the possibility of displeasure. Some variations of these models have been proposed using different notations but without changing the core meaning [72], some of which omit the dominance dimension [47].

In the Positive and Negative Affect Schedule (PANAS) [86, 87], the positive and negative affects are considered as the two primary emotional dimensions. However, these two dimensions are the result of the self-evaluation of a number of words and phrases that describe different feelings and emotions. That is: discrete emotions are rated but two dimensions are evaluated. This theory is designed to present a mood scale. Finally, positive and negative affects are mutually independent. In Figure 1, the positive (negative) dimension would comprise of positive (negative) valence, or positive arousal, or both according to the different theories.

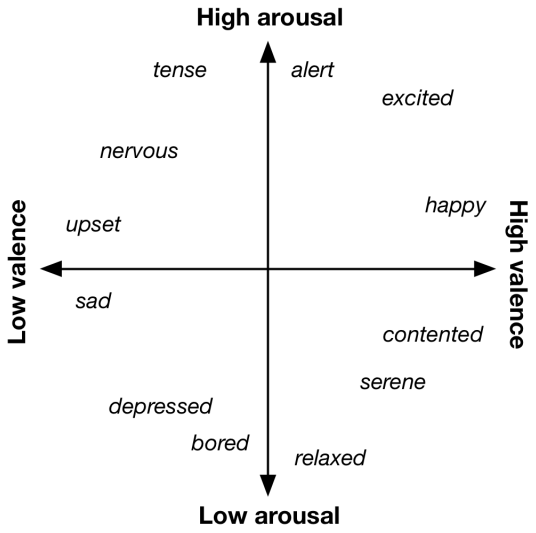

Figure 1: A PAD model of valence and arousal, and examples of related discrete affects.

We note here that several other theories exist, although they are less prominent. One example is the cube of emotion [55], which is a dimensional theory of affect that expresses affect in terms of combinations of dopamine, adrenaline, and serotonin, which intersect in eight basic (but extreme) affects, e.g. distress, interest, joy.

The Unifying Theory.

A prominent unifying theory exists as well. Russell and Barrett [74, 72, 73] have proposed the concept of *core affect* to unify the theories of emotions and moods in psychology. Core affect is "a pre-conceptual primitive process, a neurophysiological state, accessible to consciousness as a simple non-reflective feeling that is an integral blend of hedonic (valence) and arousal values: feeling good or bad, feeling lethargic or energized" [72] (p. 147). The state is accessible at a consciousness level as the simplest raw feelings, which is distinct in moods and emotions. A feeling is an assessment of one's current condition. Therefore, an affect is a very raw concept, upon which the more complex of mood and emotion is built upon. *Pride* can be thought of as feeling good about oneself. The "feeling good" is core affect and the "about oneself" is an additional (cognitive) component.

Changes in affects result from a combination of happenings, such as stressful events on the job. Sometimes the cause of the change is obvious. However, sometimes one can undergo a change in core affect without understanding the reasons. The individuals possess a limited ability to track this complex causality connection. Instead, a person makes attributions and interpretations of core affect.

Affect can be felt in relation to no obvious stimulus—in a free-floating form—as *moods* are perceived. Indeed, mood is defined as a prolonged core affect without an object, i.e. an unattributed affect.

In the core affect theory, *emotions* are episodes instead of simple objects. An emotion is a complex set of interrelated sub-events about a specific object.

The core affect theory is interesting because it unifies the previous theories, and it maintains compatibility with the

majority of the existing measurement instruments, regardless of them being about moods or emotions. Although we do not neglect moods and emotions *per se*, when adopting the core affect theory we chose to understand the states of minds of software developers at the *affective* level only, which is the foundation of moods and emotions.

Core Affect Is our Current Suggestion to Frame SE Research on Affect. However, a researcher should select the affective framework and theory that better suits the research objective and the level of details that are desirable. We provide more details in the next section. What is important is that researchers are aware of an absence of an absolute truth and of the many existing alternatives, and that they justify their choice.

3. GUIDELINES FOR PSYCHOEMPIRICAL SOFTWARE ENGINEERING

A much requested feature in our previous discussions at recent academic venues such as ISERN 2014, CHASE 2015 [6], and ICSE 2015 had been *How should one conduct research with psychological measurements?* By making sense of the hundreds of articles we reviewed on psychology and organizational behavior, we came up with a simple series of steps, listed below.

Defining a Research Objective.

As with any research activity, it is important to understand what we want to do in a study. Suppose two different, yet common scenarios with the affects of developers. They have been adapted from two of our previous studies [32, 31].

Scenario A Assessing how happy developers are generally.

Scenario B Assessing over a time frame the emotional reaction of a stimulus (e.g., employing a software tool) on developers.

Both of them require a deep understanding of the topic under study.

Theoretically Framing the Research.

Scenario A—From a comprehensive literature review, we would understand that we can call happy those developers who are in a strongly positive mood, or those who frequently have positive and meaningful experiences (see [34] for more), thus having a positive affect balance. We decide to focus on dimensions of affects, e.g. with the Positive and Negative Affect Schedule (PANAS) [86, 87], which still lets us evaluate discrete affects before the aggregated scores.

Scenario B—Suppose that, instead of asking a developer what emotions she is feeling when using a tool, we are interested in knowing how she feels in terms of more aggregated dimensions like pleasure, energy, and dominance. We focus then on the dimensional theory of affects like the one in the PAD models [75, 70, 57].

Selecting a Validated Measurement Instrument.

Scenario A—The PANAS dimensional model recommend employing the PANAS [86, 87] measurement instrument which is one of the most notable measurement instruments for affective states. However, a deeper look at the literature shows that there are several shortcomings that have been criticized for this instrument. The PANAS reportedly omits core emotions such as *bad* and *joy* while including items that are not considered emotions, like *strong*, *alert*, and *determined* [21, 54]. Another limitation has been reported in its non-consideration of the differences in desirability of emotions and feelings in various cultures [84, 54]. Furthermore, a considerable redundancy has been found in PANAS items [14, 83, 54]. PANAS has also been reported to capture only high-arousal feelings in general [21].

Recent, modern scales have been proposed to reduce the number of the PANAS scale items and to overcome some of its shortcomings. Diener [21] developed the Scale of Positive and Negative Experience (SPANE). SPANE assesses a broad range of pleasant and unpleasant emotions by asking the participants to report them in terms of their frequency during the last four weeks. It is a 12-items scale, divided into two sub-scales. Six items assess positive affective states and form the SPANE-P scale. The other six assess negative affective states and form the SPANE-N scale. The answers to the items are given on a five-point scale ranging from 1 (*very rarely or never*) to 5 (*very often or always*). For example, a score of five for the *joyful* item means that the respondent experienced this affective state *very often* or *always* during the last four weeks. The SPANE-P and SPANE-N scores are the sum of the scores given to their respective six items. Therefore, they range from 6 to 30. The two scores can be further combined by subtracting SPANE-N from SPANE-P, resulting in the Affect Balance Score (SPANE-B). SPANE-B is an indicator of the pleasant and unpleasant affective states caused by how often positive and negative affective states have been felt by the participant. SPANE-B ranges from -24 (*completely negative*) to +24 (*completely positive*). The SPANE measurement instrument has been reported to be capable of measuring positive and negative affective states regardless of their sources, arousal level or cultural context, and it captures feelings from the emotion circumplex [21, 54]. The timespan of four weeks was chosen in SPANE in order to provide a balance between the sampling adequacy of feelings and the accuracy of memory [54], and to decrease the ambiguity of people's understanding of the scale itself [21].

Scenario B—The PAD dimensional models have been implemented in several measurement instruments. One of the most notable instruments is the Affect Grid [76], which is a grid generated by intersecting the axes of valence and arousal accompanied by four discrete affects, i.e. depression-relaxation and stress-excitement, to guide the participant in pointing where the emotional reaction is located. The affect grid has been employed in SE research, e.g. in [12]. Yet, the grid was shown to have only moderate validity [43], thus other measurement instruments would be more desirable. Thus comes the Self-Assessment Manikin (SAM, [7, 48]). SAM is a pictorial, i.e. non-verbal, assessment method. SAM measures valence, arousal, and dominance associated with a person's affective reaction to an object (or a stimulus) [7]. As a picture is worth a thousand words, we reproduce SAM in figure 2. The figures of the first row range from a frown to a smile, representing the valence dimension. The second row depicts a figure showing a serene, peaceful, or passionless face to an explosive, anxious, or excited face. It represents the arousal dimension. The third row ranges from a very little, insignificant figure to a ubiquitous, pervasive figure. It represents the dominance affective dimension. As reported

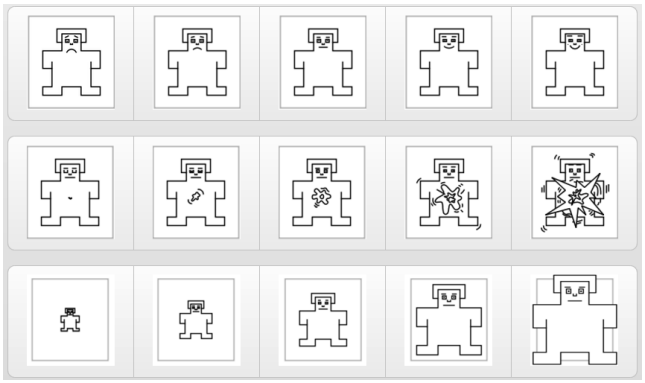

Figure 2: The Self-Assessment Manikin.

in [44], SAM has the advantage of eliminating the cognitive processes associated with verbal measures but it is still very quick and simple to use.

Considering Psychometric Properties.

As we noted in a previous paper [34], a selected measurement instrument has to possess acceptable validity and reliability properties, which are provided in psychometric studies of the measurement instrument. Psychometrics is a term, which has been misused in SE including ourselves. It is a subfield of psychology that focuses on the theory and techniques of psychological measurements. Psychometric studies deal with the design, development and especially the validation of psychological measures.

A modification to an existing measurement instrument (e.g., adding, deleting, or rewording items) often requires a new psychometric study because the reliability of a measurement instrument can be compromised. Therefore, it is not advisable to modify validated psychological measurements or models as it happened in [13].

Scenario A—The SPANE has been validated to converge with other affective states measurement instruments, including PANAS [21]. The scale provided good psychometric properties in the introductory research [21] and in numerous follow-ups, with up to twenty-one thousand participants in a single study [81, 22, 54]. Additionally, the scale proved consistency across full-time workers and students [81].

Scenario B—The SAM has been under scholarly scrutiny, as well. The original article describing SAM already reports good psychometric properties [7]. A very high correlation was found between the SAM items and those of other verbal-based measurement instruments [59, 60], including high reliability across age [2]. Therefore, SAM is one of the most reliable measurement instruments for affective reactions [44].

Administering the Measurement Instrument Correctly.

The psychometric properties of a measurement instrument in psychology are also calculated by administering the instrument in the same way in each study. This is because the instructions might influence the participants' responses. For this reason, any good measurement instrument is always accompanied with the instructions for the participants. We encourage administering a measurement instrument as it is reported in the accompanying instructions, and to further share the instructions with participants. Furthermore, the gained transparency ensures a higher reproducibility of the studies.

We strongly encourage the authors of SE studies to report the participants' instructions when publishing an article, preferably in an archived format. [1]

Scenario A—The SPANE instructions for participants are clearly stated in the original paper [21] and in the instrument itself, which is freely available. [2]

Scenario B—The SAM instructions for participants are exhaustively reported in the accompanying technical report [48].

Performing Strong Analyses.

We encourage the authors in SE to spend some time to understand whether such complex and delicate constructs require accurate analyses.

Scenario A—The SPANE scores can be considered as ordinal values or as discrete pinpoints of a continuous scale. Regression analyses on the aggregated SPANE-P, SPANE-N, and SPANE-B scores are possible given that the assumptions for linear regression are met. Otherwise, especially when groups have to be compared, the usual assumptions for employing the *t-test* or non-parametric tests should be taken into account. It is also important to report an effect size measure such as the Cohen's *d*.

Scenario B—Repeated measures within-subject that need a between subject comparison pose several issues. First, there is not a stable and shared metric for assessing the affects across persons. For example, a score of one in valence for a person may be equal to a score of three for another person. However, a participant scoring two for valence at time t and five at time $t+x$ unquestionably indicates that the participant's valence increased. As stated by Hektner [38], "*it is sensible to assume that there is a reasonable stable metric within persons*" (p. 10). In order to have comparable measurements, the raw scores of each participant are typically transformed into *z-scores* (also known as standard scores). A z-score transformation is such that a participant's mean score for a variable is zero, and scores for the same variable that lie one standard deviation above or below the mean have the value equivalent to their deviation. One observation is translated to how many standard deviations the observation itself is above or below the mean of the individual's observations. Therefore, the participants' measurements become dimensionless and comparable with each other, because the z-scores indicate how much the values are spread [50, 38].

Second, the repeated measurements often present dependencies of data at the participants' level and the time level grouped by the participant. The analysis of variance (ANOVA) family provides rANOVA as a variant for repeated measurements. However, rANOVA and general ANOVA procedures are discouraged [36] in favor of mixed-effects models, which are robust and specifically designed for repeated, within-participant longitudinal data [46, 36, 1]. A linear mixed-effects model is a linear model that contains both fixed effects and random effects [69]. The estimation of the significance

[1] For the participants' instructions in [32], see `https://dx.doi.org/10.7717/peerj.289/supp-1`. For the participants' instructions in [30, 31], see `http://dx.doi.org/10.6084/m9.figshare.796393`

[2] `http://internal.psychology.illinois.edu/~ediener/SPANE.html`

of the effects for mixed models is an open debate [4, 67]. We encourage the reader to follow our reasoning in [31] for a deeper discussion.

4. CONCLUSION

Affects—emotions and moods—are beginning to be comprehensively studied in SE, and other psychological constructs are being incorporated in related research. However, there is a risk of underusing and misusing the theory and the measurement instruments from psychology, and falling into the many misconceptions tied to such intriguing and complex research topics.

For this reason, we have proposed the term *psychoempirical software engineering* to refer to the research in SE with psychology theory and measurement. This paper described the challenge to conduct proper affect-related studies with psychology, provided a comprehensive literature review in affect theory, and proposed guidelines for conducting psychoempirical software engineering.

With this article, we hope to raise much needed awareness for better use of psychology in SE studies and to begin a sane discussion with our peers towards a more standard and sound way of conducting studies on the human and social aspects of SE.

5. ACKNOWLEDGMENTS

We are grateful for the comments received by all at the ISERN 2014 and CHASE 2015 workshops. We are also grateful to every member of the scientific community who spent time in reviewing our articles on the affect of developers, including the present one, which brought us to research in deep about this fascinating topic that deserves to be properly researched in SE.

6. REFERENCES

[1] R. Baayen, D. Davidson, and D. Bates. Mixed-effects modeling with crossed random effects for subjects and items. *Journal of Memory and Language*, 59(4):390–412, 2008.

[2] R. W. Backs, S. P. da Silva, and K. Han. A comparison of younger and older adults' self-assessment manikin ratings of affective pictures. Technical Report 4, 2007.

[3] S. G. Barsade and D. E. Gibson. Why Does Affect Matter in Organizations? *Academy of Management Perspectives*, 21(1):36–59, 2007.

[4] D. Bates. lmer, p-values and all that, 2006.

[5] C. Beedie, P. Terry, and A. Lane. Distinctions between emotion and mood. *Cognition & Emotion*, 19(6):847–878, 2005.

[6] A. Begel, R. Prikladnicki, Y. Dittrich, C. de Souza, A. Sarma, and S. Athavale. Proceedings of the 8th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE 2015). In *CHASE 2015*, 2015.

[7] M. M. Bradley and P. J. Lang. Measuring emotion: The self-assessment manikin and the semantic differential. *Journal of Behavior Therapy and Experimental Psychiatry*, 25(1):49–59, 1994.

[8] A. P. Brief and H. M. Weiss. Organizational behavior: affect in the workplace. *Annual review of psychology*, 53:279–307, 2002.

[9] M. Cabanac. What is emotion? *Behavioural Processes*, 60(2):69–83, 2002.

[10] C. Ciborra. *The Labyrinths of Information: Challenging the Wisdom of Systems*. Oxford University Press, USA, New York, USA, 1 edition, 2002.

[11] G. L. Clore and A. Ortony. Appraisal Theories: How Cognition Shapes Affect into Emotion. In M. Lewis, J. M. Haviland-Jones, and L. Feldman-Barrett, editors, *Handbook of Emotions*, chapter 39, pages 628–642. The Guilford Press, New York, USA, 3 edition, 2008.

[12] R. Colomo-Palacios, C. Casado-Lumbreras, P. Soto-Acosta, and A. García-Crespo. Using the affect grid to measure emotions in software requirements engineering. *Journal of Universal Computer Science*, 17(9):1281–1298, 2011.

[13] R. Colomo-Palacios, C. Casado-Lumbreras, P. Soto-Acosta, and A. García-Crespo. Decisions in software development projects management. An exploratory study. *Behaviour and Information Technology*, 32(11):1077–1085, 2013.

[14] J. R. Crawford and J. D. Henry. The positive and negative affect schedule (PANAS): construct validity, measurement properties and normative data in a large non-clinical sample. *The British journal of clinical psychology / the British Psychological Society*, 43(Pt 3):245–65, 2004.

[15] M. Csikszentmihalyi. *Finding flow: The psychology of engagement with everyday life*, volume 30. Basic Books, New York, USA, 1 edition, 1997.

[16] Daniel Graziotin. The Dynamics of Creativity in Software Development. In *Doctoral Symposium of the 14th International Conference on Product-Focused Software Process Improvement (PROFES 2013)*, pages 1–6, Paphos, Cyprus, 2013. `http://dx.doi.org/10.6084/m9.figshare.703568`.

[17] C. K. W. De Dreu, B. a. Nijstad, M. N. Bechtoldt, and M. Baas. Group creativity and innovation: A motivated information processing perspective. *Psychology of Aesthetics, Creativity, and the Arts*, 5(1):81–89, 2011.

[18] C. DeLancey. Basic Moods. *Philosophical Psychology*, 19(4):527–538, 2006.

[19] P. J. Denning. Moods. *Communications of the ACM*, 55(12):33, 2012.

[20] P. Dewan. Towards Emotion-Based Collaborative Software Engineering. In *ICSE*, 2015.

[21] E. Diener, D. Wirtz, W. Tov, C. Kim-Prieto, D. Choi, S. Oishi, and R. Biswas-Diener. New Well-being Measures: Short Scales to Assess Flourishing and Positive and Negative Feelings. *Social Indicators Research*, 97(2):143–156, 2009.

[22] T. Dogan, T. Totan, and F. Sapmaz. The Role Of Self-esteem, Psychological Well-being, Emotional Self-efficacy, And Affect Balance On Happiness: A Path Model. *European Scientific Journal*, 9(20):31–42, 2013.

[23] P. Ekman. Universals and cultural differences in facial expressions of emotion. In *Nebraska Symposium on Motivation*, pages 207–283, 1971.

[24] P. Ekman. An argument for basic emotions. *Cognition & Emotion*, 6(3):169–200, 1992.

[25] R. Feldt, L. Angelis, R. Torkar, and M. Samuelsson. Links between the personalities, views and attitudes of

software engineers. *Information and Software Technology*, 52(6):611–624, 2010.

[26] G. Fischer. Cognitive View of Reuse and Redesign. *IEEE Software*, 4(4):60–72, 1987.

[27] C. D. Fisher and N. M. Ashkanasy. The emerging role of emotions in work life: an introduction. *Journal of Organizational Behavior*, 21(2):123–129, 2000.

[28] D. Ford and C. Parnin. Exploring Causes of Frustration for Software Developers. In *2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 115–116, 2015.

[29] C. França, H. Sharp, and F. Q. B. da Silva. Motivated software engineers are engaged and focused, while satisfied ones are happy. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '14*, pages 1–8, New York, USA, 2014. ACM Press.

[30] D. Graziotin, X. Wang, and P. Abrahamsson. Are happy developers more productive? The correlation of affective states of software developers and their self-assessed productivity. In *14th International Conference of Product-Focused Software Process Improvement. LNCS*, volume 7983, pages 50–64, 2013.

[31] D. Graziotin, X. Wang, and P. Abrahamsson. Do feelings matter? On the correlation of affects and the self-assessed productivity in software engineering. *Journal of Software: Evolution and Process*, Early View:1–21, 2014. `http://doi.wiley.com/10.1002/smr.1673`.

[32] D. Graziotin, X. Wang, and P. Abrahamsson. Happy software developers solve problems better: Psychological measurements in empirical software engineering. *PeerJ*, 2:e289, 2014.

[33] D. Graziotin, X. Wang, and P. Abrahamsson. Software developers, moods, emotions, and performance. *IEEE Software*, 31(4):24–27, 2014.

[34] D. Graziotin, X. Wang, and P. Abrahamsson. The Affect of Software Developers: Common Misconceptions and Measurements. In *2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 123–124, Firenze, Italy, 2015. IEEE Computer Society.

[35] D. Graziotin, X. Wang, P. Abrahamsson, and A. Jedlitschka. Psychoempirical Software Engineering. In *2014 Annual Meeting of the International Software Engineering Research Network (ISERN 2014)*, Turin, Italy, 2014. `http://dx.doi.org/10.6084/m9.figshare.1169833`.

[36] R. Gueorguieva and J. H. Krystal. Move over ANOVA: progress in analyzing repeated-measures data and its reflection in papers published in the Archives of General Psychiatry. *Archives of General Psychiatry*, 61(3):310–7, 2004.

[37] L. Haaranen, P. Ihantola, J. Sorva, and A. Vihavainen. In Search of the Emotional Design Effect in Programming. In *37th International Conference on Software Engineering (ICSE 2015)*, 2015.

[38] J. M. Hektner, J. A. Schmidt, and M. Csikszentmihalyi. *Experience Sampling Method: Measuring the Quality of Everyday Life*. Sage Publications Inc, 2007.

[39] M. Huang. The theory of emotions in marketing. *Journal of Business and Psychology*, 16(2):239–247, 2001.

[40] C. E. Izard. *Human Emotions*. Plenum Press, New York, USA, 1 edition, 1977.

[41] D. Kahneman. Maps of bounded rationality: Psychology for behavioral economics, 2003.

[42] I. A. Khan, W. Brinkman, and R. M. Hierons. Do moods affect programmers' debug performance? *Cognition, Technology & Work*, 13(4):245–258, 2010.

[43] W. D. Killgore. The Affect Grid: a moderately valid, nonspecific measure of pleasure and arousal. *Psychological reports*, 83(2):639–642, 1998.

[44] J. Kim, J. A. Geason, C. Woo, and J. D. Morris. The Power of Affect: Predicting Intention. *Journal of Advertising Research*, 42(3):7–17, 2002.

[45] P. R. Kleinginna and A. M. Kleinginna. A categorized list of emotion definitions, with suggestions for a consensual definition. *Motivation and Emotion*, 5:345–379, 1981.

[46] N. M. Laird and J. H. Ware. Random-Effects Models for Longitudinal Data. *Biometrics*, 38(4):963–974, 1982.

[47] R. D. Lane, P. M.-L. Chua, and R. J. Dolan. Common effects of emotional valence, arousal and attention on neural activation during visual processing of pictures. *Neuropsychologia*, 37(9):989–997, 1999.

[48] P. J. Lang, M. M. Bradley, and B. N. Cuthbert. International affective picture system (IAPS): Technical manual and affective ratings. *Gainesville FL NIMH Center for the study of emotion and attention University of Florida*, pages Technical Report A–6, 1999.

[49] P. J. Lang, M. K. Greenwald, M. M. Bradley, and A. O. Hamm. Looking at pictures: Affective, facial, visceral, and behavioral reactions. *Psychophysiology*, 30(3):261–273, 1993.

[50] R. Larson and M. Csikszentmihalyi. The experience sampling method. *New Directions for Methodology of Social and Behavioral Science*, 15(15):41–56, 1983.

[51] P. Lenberg, R. Feldt, and L.-G. Wallgren. Towards a behavioral software engineering. In *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering - CHASE 2014*, pages 48–55, New York, USA, 2014. ACM Press.

[52] P. Lenberg, R. Feldt, and L. G. Wallgren. Behavioral software engineering: A definition and systematic literature review. *Journal of Systems and Software*, page In Press, 2015.

[53] K. Lewin. *A dynamic theory of personality*, volume 2008. McGraw-Hill, New York, USA, 1935.

[54] F. Li, X. Bai, and Y. Wang. The Scale of Positive and Negative Experience (SPANE): psychometric properties and normative data in a large Chinese sample. *PloS one*, 8(4):e61137, 2013.

[55] H. Lövheim. A new three-dimensional model for emotions and monoamine neurotransmitters. *Medical Hypotheses*, 78(2):341–348, 2012.

[56] A. Medinilla. Process Kaizen Agile Kaizen. *Process Kaizen: Managing Continuous Improvement Far Beyond Retrospectives (Part 1)*, pages 85–119, 2014.

[57] A. Mehrabian. Pleasure-arousal-dominance: A general framework for describing and measuring individual differences in Temperament. *Current Psychology*, 14(4):261–292, 1996.

[58] M. Minsky. A Framework for Representing Emotional States. In M. Lewis, J. M. Haviland-Jones, and L. Feldman-Barrett, editors, *The Handbook of Emotions*, chapter 38, pages 618–627. The Guilford Press, New York, USA, 3 edition, 2008.

[59] J. Morris and C. Waine. Managing the creative effort: Pre-production and post-production measures of emotional response. In *Proceedings of the 1993 Conference of the American Academy of Advertising*, pages 158–156, 1993.

[60] J. D. Morris. SAM: The Self-Assessment Maniking - An Efficient Cross-Cultural Measurement of Emotional Response. *Journal of Adverstising Research*, 35(6):63–68, 1995.

[61] P. M. Muchinsky. Emotions in the workplace: the neglect of organizational behavior. *Journal of Organizational Behavior*, 21(7):801–805, 2000.

[62] S. C. Müller and T. Fritz. Stuck and Frustrated or In Flow and Happy : Sensing Developers ' Emotions and Progress. In *37th International Conference on Software Engineering (ICSE 2015)*, 2015.

[63] A. Murgia, P. Tourani, B. Adams, and M. Ortu. Do developers feel emotions? an exploratory analysis of emotions in software artifacts. In *11th Working Conference on Mining Software Repositories*, pages 262–271, New York, USA, 2014. ACM Press.

[64] A. Ortony, G. L. Clore, and A. Collins. *The Cognitive Structure of Emotions*. Cambridge University Press, New York, USA, 1 edition, 1990.

[65] B. Parkinson, R. Briner, Reynolds S., and P. Totterdell. *Changing moods: The psychology of mood and mood regulation*. Addison-Wesley Longman, London, 1 edition, 1996.

[66] R. Plutchik and H. Kellerman. *Emotion, theory, research, and experience*, volume 1. Academic Press, London, 1980.

[67] R Community. Conservative ANOVA tables in lmer, 2006.

[68] R. Reisenzein. What is a definition of emotion? And are emotions mental-behavioral processes? *Social Science Information*, 46(3):424–428, 2007.

[69] G. K. Robinson. That BLUP is a Good Thing: The Estimation of Random Effects. *Statistical Science*, 6(1):15–32, 1991.

[70] J. A. Russell. A circumplex model of affect. *Journal of Personality and Social Psychology*, 39(6):1161–1178, 1980.

[71] J. A. Russell. Culture and the categorization of emotions. *Psychological Bulletin*, 110(3):426–450, 1991.

[72] J. A. Russell. Core affect and the psychological construction of emotion. *Psychological Review*, 110(1):145–172, 2003.

[73] J. a. Russell. Emotion, core affect, and psychological construction. *Cognition & Emotion*, 23(7):1259–1283, 2009.

[74] J. A. Russell and L. F. Barrett. Core affect, prototypical emotional episodes, and other things called emotion: dissecting the elephant. *Journal of personality and social psychology*, 76(5):805–819, 1999.

[75] J. A. Russell and A. Mehrabian. Evidence for a three-factor theory of emotions. *Journal of Research in Personality*, 11(3):273–294, 1977.

[76] J. a. Russell, A. Weiss, and G. a. Mendelsohn. Affect Grid: A single-item scale of pleasure and arousal., 1989.

[77] D. Sato, D. Bassi, M. Bravo, A. Goldman, and F. Kon. Experiences tracking agile projects: an empirical study. *Journal of the Brazilian Computer Society*, 12(3):45–64, 2006.

[78] N. Schwarz. Feelings as information: informational and motivational functions of affective states. In E. T. Higgins and R. M. Sorrentino, editors, *Handbook of Motivation and Cognition: Foundations of Social Behavior*, number 89, chapter 15, pages 527–561. Guilford Press, New York, NY, US, 1990.

[79] N. Schwarz and G. L. Clore. Mood, misattribution, and judgments of well-being: Informative and directive functions of affective states. *Journal of Personality and Social Psychology*, 45(3):513–523, 1983.

[80] P. Shaver, J. Schwartz, D. Kirson, and C. O'Connor. Emotion knowledge: Further exploration of a prototype approach. *Journal of Personality and Social Psychology*, 52(6):1061–1086, 1987.

[81] A. J. Silva and A. Caetano. Validation of the Flourishing Scale and Scale of Positive and Negative Experience in Portugal. *Social Indicators Research*, 110(2):469–478, 2011.

[82] D. Sjoberg, T. Dybå, B. Anda, and J. Hannay. Building theories in software engineering. *Guide to Advanced Empirical Software Engineering*, 1(1):312–336, 2008.

[83] E. R. Thompson. Development and Validation of an Internationally Reliable Short-Form of the Positive and Negative Affect Schedule (PANAS). *Journal of Cross-Cultural Psychology*, 38(2):227–242, 2007.

[84] J. L. Tsai, B. Knutson, and H. H. Fung. Cultural variation in affect valuation. *Journal of personality and social psychology*, 90(2):288–307, 2006.

[85] G. R. VandenBos. *APA dictionary of clinical psychology*. American Psychological Association, 2013.

[86] D. Watson, L. A. Clark, and A. Tellegan. The Positive and Negative Affect Schedule. *Journal of Personality*, 8(6):1988, 1988.

[87] D. Watson, L. A. Clark, and A. Tellegen. Development and validation of brief measures of positive and negative affect: The PANAS scales. *Journal of Personality and Social Psychology*, 54(6):1063–1070, 1988.

[88] J. Wegge, R. V. Dick, G. K. Fisher, M. a. West, and J. F. Dawson. A Test of Basic Assumptions of Affective Events Theory (AET) in Call Centre Work. *British Journal of Management*, 17(3):237–254, 2006.

[89] H. Weiss and R. Cropanzano. Affective events theory: A theoretical discussion of the structure, causes and consequences of affective experiences at work. *Research in Organizational Behavior*, 18(1):1–74, 1996.

[90] M. Zajenkowski, E. Goryńska, and M. Winiewski. Variability of the relationship between personality and mood. *Personality and Individual Differences*, 52(7):858–861, 2012.

The Challenges of Sentiment Detection in the Social Programmer Ecosystem

Nicole Novielli, Fabio Calefato, Filippo Lanubile
University of Bari
Dipartimento di Informatica
Bari, Italy
{nicole.novielli, fabio.calefato, filippo.lanubile}@uniba.it

ABSTRACT
A recent research trend has emerged to study the role of affect in in the social programmer ecosystem, by applying sentiment analysis to the content available in sites such as GitHub and Stack Overflow. In this paper, we aim at assessing the suitability of a state-of-the-art sentiment analysis tool, already applied in social computing, for detecting affective expressions in Stack Overflow. We also aim at verifying the construct validity of choosing sentiment polarity and strength as an appropriate way to operationalize affective states in empirical studies on Stack Overflow. Finally, we underline the need to overcome the limitations induced by domain-dependent use of lexicon that may produce unreliable results.

Categories and Subject Descriptors
H.1.2 [**User/Machine Systems**]: Human factors

General Terms
Human Factors.

Keywords
Online Q&A, Technical Forum, Sentiment Analysis, Stack Overflow, Social Programmer, Social Software Engineering

1. INTRODUCTION
Software engineering involves a large amount of social interaction, as programmers often need to cooperate with others, whether directly or indirectly. However, we have become fully aware of the importance of social aspects in software engineering activities only over the last decade. In fact, it was not until the recent diffusion and massive adoption of social media that we could witness the rise of the "social programmer" [41] and the surrounding ecosystem [42].

Social media has deeply influenced the design of software development-oriented tools such as GitHub (i.e., a social coding site) and Stack Overflow (i.e., a community-based question answering site) [43]. Stack Overflow, in particular, is an example of an online community where social programmers do networking by reading and answering others' questions, thus participating in the creation and diffusion of crowdsourced documentation. In our previous work, we argued and proved that among the non-technical factors, which can influence the members of online communities, the emotional style of a technical contribution does affect its probability of success [29], [9]. More specifically, our effort is to understand how expressing affective states in Stack Overflow influences the probability for askers of eliciting an accepted answer and the probability for answerers of having an answer accepted.

Our research follows a recent trend that has emerged to study the role of affect in social computing. For example, Kucuktunc et al. [19] performed a large-scale sentiment analysis study on Yahoo! Answers to assess the impact of the semantic orientation of a post on its perceived quality. Althoff et al. [1] found that expressing gratitude in a question is positively correlated with success of altruistic requests in Reddit.com. Guzman et al. [17] perform sentiment analysis of commit comments in GitHub and demonstrate that a correlation exists between emotions and other factors such as the programming language used in a project, the geographical distribution of the team and the day of the week. Similarly, Guzman and Bruegge [16] used a sentiment analysis tools for detecting the polarity, i.e., the positive or negative semantic orientation of a text, to investigate the role of emotional awareness in software development teams.

What these studies have in common is that they applied sentiment analysis techniques to crowd-generated content relying on polarity as the only dimension to operationalize affect. However, polarity is only one of the possible dimensions of affect, which could be also modeled in terms of its duration, activation, cognitive triggers, and specificity [11]. Still, polarity is the most used dimension because of its ease of measurement and the availability of open source and robust analysis tools. In this paper, we argue that polarity, if employed alone, is insufficient for detecting the sentiments of programmers in a reliable manner. Furthermore, we highlight and discuss the challenges existing when sentiment analysis techniques are employed to assess the affective load of text containing technical lexicon, as typical in the social programmer ecosystem.

The remainder of the paper is structured as follows. In Section 2, we first provide an overview of detecting affective states from text, including a state-of-the-art in the field of sentiment analysis. Then, in Section 3, we perform a qualitative analysis to show the limits of only using polarity to measure the sentiment expressed in questions and answers in Stack Overflow. The findings from our analysis are then discussed in Section 4, where we also outline the future research directions.

SSE'15, September 1, 2015, Bergamo, Italy
© 2015 ACM. 978-1-4503-3818-9/15/09...$15.00
http://dx.doi.org/10.1145/2804381.2804387

2. MINING AFFECTIVE STATES FROM TEXT

2.1 Affect Modeling Theories

Nowadays, affective computing is an established discipline [35] and sensing affective states from text is now regarded as fundamental in several domains, from human-computer interaction [7][21][30] to software engineering [8][26][16].

Affective states vary in their degree of stability and in their 'object-oriented specificity' [11], ranging from personality traits to emotions. *Personality traits* are long-standing, organized sets of characteristics of a person that uniquely influences cognitions, motivations and behaviors. *Emotions* are transient and typically complex, episodic, dynamic and structured events, which involve perceptions, thoughts, feelings, bodily changes, and personal dispositions to experience further emotional episodes. Emotions are episodic and dynamic in that, over time, the elements can come and go, depending on all sorts of factors. Other states, such as *interpersonal stance* or *attitudes* are in a middle of this scale: they are initially triggered by individual characteristics like personality, social role, and others but may vary, in valence (i.e. positive vs. negative) and intensity (i.e. low arousal vs. high arousal) by episodes occurring during communication.

Mining affective states from text involves, on one hand, to model them according to bi-dimensional models representing the affect polarity or valence and its level of activation or intensity; on the other hand, some studies explicitly deal with discrete emotion labeling of text, by looking for linguistic cues of specific affective states. Psychologists worked at decoding emotions for decades, by focusing on two main questions: (i) how can emotions be classified? (ii) What is their functioning?, i.e., How are they triggered? How do they affect behavior? Which is the role played by cognition? Two points of view prevailed: the first one assumes that a limited set of basic emotions exists, while the second one consider emotions as a continuous function of one or more dimensions. It is the case of the 'circumplex model' of affect [40], which models emotions along a bi-dimensional representation schema, including valence (pleasant vs. unpleasant) and arousal (activation vs. deactivation) of emotions. Conversely, theories following the discrete trend agree on the idea that a limited set of basic emotions exists, although consensus about the nature and the number of these basic emotions has not been reached. Ekman defines a basic emotion as having specific feelings, universal signals and corresponding physiological changes [12]; Lazarus describes nine negative (Anger, Anxiety, Guilt, Shame, Sadness, Envy, Jealousy, Disgust) and six positive (Joy, Pride, Love, Relief, Hope, Compassion) emotions, with their appraisal patterns: positive emotions are triggered if the situation is congruent with one of the individual's goals; otherwise, negative emotions are triggered [20]; Plutchik defines discrete emotions as corresponding to specific adaptive processes: reproduction, safety, etc. [36]. The Plutchik's model identifies eight primary emotions (i.e., anger, fear, sadness, disgust, surprise, anticipation, trust, and joy), graphically depicted as a wheel (see Figure 1). Opposite affective states, with respect to both valence and intensity, are placed opposite of each other using complementary colors. Strength of emotions increases towards the center of the model, with low activation affective states lying next to the circumference.

During the last decade, in computational linguistics several approaches have been investigated for mining affect from text. With respect to the specific goals addressed and to the adopted theory, researchers model and detect affect at a different granularity. Several studies refer to discrete emotion categorization. For example, Liu et al. [23] propose a method based on large-scale real-world knowledge about the way people usually make appraisals of everyday situations. The approach exploits generic knowledge basis of commonsense to identify the six Ekman's basic emotional states (happy, sad, angry, fearful, disgusted, and surprised) through the text analysis. Neviarouskaya et al. [27] use a rule-based approach that includes consideration of the deep syntactic structures for emotion computation in the text. Experiments are performed on different corpora to identify nine emotional labels (plus the neutral one). Compared with other state-of-the-art techniques, the method shows promising results in fine-grained emotion recognition. Other studies, rather focuses on the valence (i.e. the positive or negative orientation) of affective states conveyed in natural language interactions. It is the case of Litman et al. [22], who defined an annotation scheme to label emotions in tutoring dialogs along a linear scale (negative, neutral, positive) in order to detect students' boredom and frustration. Analogously, Batliner et al. [7], define a method for automatically detecting emotionally critical phases in dialogues with customers of an automatic call-center, in order to enhance customers' satisfaction by adapting the interaction accordingly. Le Tallec et al. [21] studied how to classify emotions in speech by considering the language of hospitalized children interacting with companion robots. Linguistic clues are considered, achieving good results in detecting emotional valence of utterances. Novielli et al. [30], exploit linguistic cues to detect cold vs. warm social attitude of users toward an Embodied Conversational Agents in the domain of simulation of persuasion dialogues.

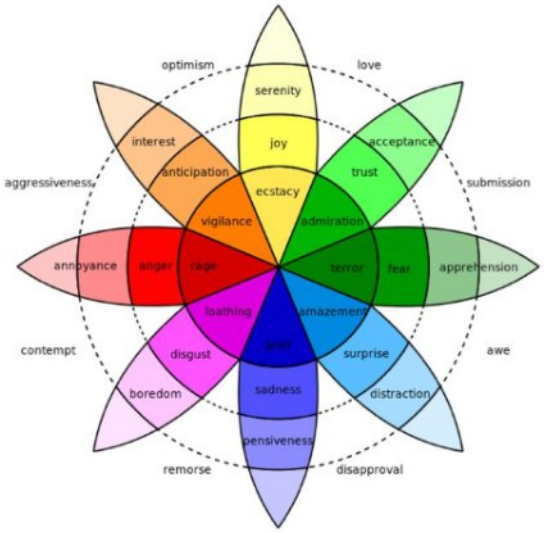

Figure 1. The Plutchik Wheel [36]

2.2 State of the Art on Sentiment Analysis

As far as affect polarity is concerned, sentiment analysis provides researchers with a suite of methods and linguistic resources, which can be exploited for recognizing the semantic orientation of texts. Sentiment analysis is the study of the subjectivity and polarity of a text [33]. More recently, researcher in this field started to deal also with sentiment strength detection, obtaining promising results [47]. Sentiment analysis techniques have been successfully applied to the problem of detecting the valence of affective states conveyed by a text [16], for modeling socio-

economic phenomena [32], and to automatically analyze text for opinion mining purposes [18][33].

2.2.1 Approaches

Traditional approaches to sentiment analysis treat the subjectivity and polarity detection as text classification problems and exploit machine learning algorithms for extracting features to train supervised classifiers on human-annotated corpora. The features employed are typically words (i.e., tokens, stems, lemmata) and part-of-speech tags, also combined in n-grams, that is sequences of *n* contiguous words. Such approaches mainly rely on state-of-the-art machine learning algorithms, such as Support Vector Machines [48], and might also be improved by performing intelligent feature selection [38]. With the worldwide diffusion of social media, a huge amount of textual data has been made available, thus attracting the interest of researchers in this domain [39]. Sentiment analysis on such informal texts poses new challenges due to the presence of slang, misspelled words, hashtags, and links, thus inducing researchers to define novel approaches that include consideration of micro-blogging features [6][25].

However, supervised approaches present the main drawback of being highly domain-dependent. This means that classification models are very likely to perform poorly outside the domain they were trained on [15]. In fact, when training classification models, it is very likely to include consideration of terms that associate with sentiment because of the domain. It is the case of political debates, where names of countries afflicted by wars might be associated to negative sentiments; analogous problems might be observed for the technology domain, where killer features usually referred in positive customers' review usually become obsolete in relatively short periods of time. Such terms are usually referred by researchers as *indirect affective words* to distinguish them from *direct* ones [45]. Indeed, according to the emotion theory defined by Clore et al. [10], it is possible to distinguish between words that directly refer to emotional states (e.g., 'fear', 'joy', 'cheerful', 'sad') and those having only an indirect reference to an emotional state, depending on the context (e.g., the words which indicates emotional causes such as 'killer' or 'monster' or emotional responses to an event such as 'cry' or 'laugh').

To overcome these limitations, lexical approaches are adopted, which exploit the prior sentiment polarity of words in a text, based on lexical resources, i.e. large lexicons of words annotated with their prior polarity (i.e. positive or negative semantic orientation of the word). The overall sentiment of a text is computed based on the prior polarity of the words composing it [28][46] as well as on their contextual polarity (i.e. the polarity conveyed by a word with respect to its context of usage) [4][49]. In fact, words with negative prior polarity can be used to express positive sentiment (as in '*I feel so sorry for you*' where a positive attitude towards the interlocutor is expressed) or even in a neutral context. Furthermore, the effect of contextual valence shifter [37], such as intensifiers or negation, needs to be taken into account since they might intensify, mitigate or even invert the polarity of the word they are associated with. Therefore, lexicon-based approaches are usually integrated with other knowledge such as semantic rules [27][46] or features specific of the communication medium, as for example emoticon lists [47].

2.2.2 Linguistic Resources

In this section, we provide an overview on the state-of-the-art linguistic resources for sentiment analysis and affect mining from text.

Sentiment lexicons are basically organized as lists of words with scores indicating their prior polarity and, in some cases, also the sentiment strength. These lexicons can be differentiated based on how they represent the information about prior polarity of words.

The Bing Liu Lexicon of Opinion Words [18] is a manually created lexicon of about 6800 words, built upon e-commerce customer reviews. The annotation is binary and simply states if a word expresses positive or negative sentiment.

Similarly, the MPQA Lexicon [49] provides a broader list of positive and negative terms (about 8k), also including information about the strength of the sentiment conveyed (i.e. a word can be categorized as either 'weakly' or 'strongly' subjective.The part of speech is also reported, since different prior polarity annotations can be assigned to the same word, based on the role it plays in the discourse. The MPQA Lexicon is part of the OpinionFinder[1], a system for automatic identification of subjective sentences in documents.

The NRC Emotion Lexicon [24] contains about 14K entries consisting of words for which annotation is provided based on both polarity and a set of discrete emotion labels (i.e. anger, fear, sadness, disgust, surprise, anticipation, trust, and joy). The lexicon has been created starting from an annotation of word-sense pairs. Each word-sense pair is annotated by at least three annotators and the final word-level lexicon was created by considering the union of the emotion labels provided for all the senses of a word.

Similarly, the NRC Hashtag Sentiment Lexicon and the Sentiment140 Lexicon provide lists of words with their sentiment association score, calculated as pointwise mutual information with respect to collections of positive and negative tweets [25]. Unlike the previous ones, these two lexica have been created exploiting a completely automatic training procedure based on a corpus of English tweets annotated using positive and negative hashtags and emoticons as 'noisy' labels.

Another widely used lexicon designed on purpose for serving sentiment analysis tasks is SentiWordNet 3.0 [13]. SentiWordNet extends Word-Net [14] by associating positive, negative and objective scores to each synset (i.e., set of synonyms), where the three scores sum up to 1. A word can receive multiple polarity scores if it occurs in more than one synset (see example scores for the word 'good' reported in Table 1). Thanks to the availability of explicit objective scores, additional features can be computed to model the presence of neutral terms, as reported in [6].

WordNet Affect [44] also extends the WordNet database with affective labels (a-labels) for annotation of synsets. One or more a-labels may be assigned to a synset. The resource also includes a-labels representing moods, situations eliciting emotions, or emotional responses (see examples in Table 2).

In this review, we include the Linguistic Inquiry and Word Count (LIWC) taxonomy, even if it has been developed in the scope of broader psycholinguistic research [34] without being explicitly designed for sentiment analysis. LIWC organizes words into psychologically meaningful categories based on the assumption that words and language reflect most part of cognitive and emotional phenomena involved in communication. Previous research has shown how the language use varies with respect to the communicative intention, thus making possible to distinguish between objective and subjective statements as well as between agreement and disagreement expressions [31]. In fact, among the

[1] http://mpqa.cs.pitt.edu/opinionfinder

word classes included in the taxonomy, LIWC provides linguistic categories that draw distinctions between negative and positive emotion lexicon, which research could use to derive word-count metrics to assess the affective load of a text.

Table 1. SentiWordNet scores for the word 'good'

Scores	Sense ID and gloss
as an adjective	
Positive = 0.75 Negative = 0 Objective = 0.25	good#1 Having desirable or positive qualities especially those suitable for a thing specified (as in 'a good joke')
Positive = 0 Negative = 0 Objective = 1	good#2 having the normally expected amount (as in 'gives good measure')
Positive = 1 Negative = 0 Objective = 0	good#6 agreeable or pleasing (as in 'we all had a good time')
as a noun	
Positive = 0.5 Negative = 0 Objective = 0.5	good#1 benefit (as in 'for your own good')
Positive = 0.875 Negative = 0 Objective = 0.125	good#2 moral excellence or admirableness (as in 'there is good in people')
Positive = 0 Negative = 0 Objective = 1	good#4 commodity, article of commerce

Table 2. A-Labels in WordNet Affect with Examples

A-label	Example of Synsets
EMOTION	noun 'anger', verb 'fear'
MOOD	noun 'animosity', adjective 'fear'
TRAIT	noun 'aggressiveness', adj. 'competitive'
COGNITIVE State	noun 'confusion', adj. 'dazed'
PHYSICAL State	noun 'illness'
HEDONIC Signal	noun 'hurt', noun 'suffering'
Emotion-eliciting SITUATION	noun 'awkwardness'
Emotional RESPONSE	noun 'cold sweat', verb 'tremble'
BEHAVIOR	noun 'offense', adj. 'inhibited'
ATTITUDE	noun 'intolerance', noun 'defensive'
SENSATION	noun 'coldness', noun 'feel'

3. ANALYSIS OF AFFECTIVE STATES IN STACK OVERFLOW

3.1 Dataset and Instrumentation

We built the dataset for our analysis starting from the official Stack Overflow dumps.[2] As for the questions and askers' comments, they were extracted from the Stack Overflow dump, updated on May 2014. Instead, the answers and answerers' comments were extracted from the official Stack Overflow dump of September 2014, as described in [9]. For each question, answer and comment in our dataset, we annotated the sentiment score, measured in terms of text polarity and its strength. Then, we built our final dataset for the qualitative analysis by opportunistically choosing the cases with the highest sentiment score.

In order to measure the sentiment load of a contribution, we look for affective lexicon in the body of questions, answers and comments. Specifically, we measure the overall positive/negative polarity of a text as well as the sentiment strength. We deliberately choose to capture the *sentiment* of text, that is, its positive/negative semantic orientation of the text. Sentiment is calculated for each post in our dataset using SentiStrength[3], a state of the art tool already employed in previous research on sentiment analysis in social computing [1][17][19], which has been designed to overcome the limitations due to domain-dependence of supervised approaches. SentiStrength is a lexicon-based classifier that exploits rules and a lexicon built by combining entries from different linguistic resources. Furthermore, it has been validated for the social web and therefore it is robust for the analysis of informal text including web jargon (such as emoticons or abbreviations) [47].

Based on the assumption that a sentence can convey mixed sentiment, SentiStrength outputs both positive and negative sentiment scores for any input text written in English. It assigns the overall positive and negative scores to a text by considering the maximum among all the sentence scores**.** In SentiStrength, positive sentiment scores range from ±1 (neutral) to +5 (extremely positive/negative). In our analysis, we adjust the sentiment score and map them into the ±[0,4] interval, with zero indicating the absence of positive or negative sentiment ('neutral' texts). These metrics represent the overall positive/negative polarity and strength of the sentiment conveyed by a post, whether a question, an answer or a comment. For each post, we issue both its positive and negative score. When both are null, the post is considered as *neutral* (no sentiment is expressed through the lexicon employed). A few examples are provided in Table 3.

We analyzed the top 100 questions and follow-up askers' comments with the highest positive and negative scores (400 cases overall). Analogously, we analyzed the 100 top answers and follow-up answerers' comments with the highest positive and negatives score (400 additional cases).

[2] https://archive.org/details/stackexchange

[3] http://sentistrength.wlv.ac.uk

Table 3. Examples of sentiment expression in questions in Stack Overflow with associated sentiment scores

"*I have very simple and **stupid trouble** […] I'm **pretty confused**, explain please, what is wrong*?" Positive Score: 0; Negative Score: 1
*"**Thank you**, that was **really helpful**"* Positive score: 3; Negative Score: 0

3.2 Affect in Questions and Askers' Comments

As for questions with an overall negative polarity, the analysis reveals that the askers generally express a negative emotion associated with their technical issue or report a negative opinion. In the first case, information seekers mainly express their frustration for not being able to solve a problem as in the following examples:

- *"I am not sure what I did in a previous life to warrant this, it must have been bad! I am getting buried in a world of xml […]"*
- *"I recently got stuck on an odd problem".*

Conversely, negative opinions mainly refer to a preference or evaluation about a technical issue, as in these examples.

- *"They use it to clean up connections, which is really scary"*
- *"It is very painful to add multiple tickets to Trac",*
- *"I find this incredibly annoying with Dreamweaver".*

As for negative comments added by the asker, opinions about tools and resources are the main reason for use of negative affective lexicons as in

- *"I really hate those properties panels that don't look the same whether they are VB/C# winform/web. This sucks!"*

Again, we notice information seekers expressing their distress:

- *"This is driving me nutz :-("*

frustration:

- *"Unfortunately Xcode doesn't use CodeSense for editing files outside a project; which is incredibly frustrating."*
- *"I still get 400 bad request page! :(("*

and fear:

- *"there's no way to do this I'm afraid :("*

Cases of humor are also detected:

- *"Haha that comment made me laugh my heart out. Makes me kind of proud of how horrible my code can be!"*
- *"I would also like to add that I do find humor in these sorts of problems. This is because the only other option is to become horribly horribly angry. And who wants that?!"*

as well as a few instances of offending sarcastic comments:

- *"Do u know [the] answer ? If not .... then u might know how hard is homework. Ignorance is bliss".*

As for questions with positive sentiment score, we do not observe actual emotion reporting. On the contrary, we rather find opinions, as in the following examples:

- *"Given past frustrations with Win32 inextensibility, this seems like a good thing"*
- *"[…] I'm checkin out what WCF has to offer. It seems very flexible and a great next step up from".*
- *"I'm loving the built-in Clang!! Kudos to Xcode team!").*

Furthermore, we observe a considerable use of affective lexicon for politeness expressions. This is typical of so-called '*behabitives*' [5], speech acts in which no real feelings are expressed but still emotional words are employed to convey other communicative intentions. A typical use found in our Stack Overflow dataset is in expressing gratitude in advance towards potential helpers, as in the following examples:

- *"Any help is hugely appreciated!"*
- *"Any example would be super magnificent!"*

Evidence of analogous use of positive lexicon is found in comments. We found cases where no true feelings experienced by the asker are reported. Rather, gratitude is expressed (e.g., *"Wow, great! Thanks"*) as well as appreciation for the solution provided in the previous comments by other Stack Overflow users, e.g.:

- *"Excellent link! You should have posted it as an answer :P"*
- *"Excellent! Thanks for the info."*

Technical opinions are also reported in a few cases:

- *"As modeler we use Activity Designer. Excellent tool!"*

3.3 Affect in Answers and Answerers' Comments

As for answers, we observe that a negative lexicon is used for expressing actual emotions in the great majority of cases. However, the negative polarity of these answers does not involve a negative judgment of the asker but it rather results in either an attempt of showing empathy towards the asker as in the following examples:

- *"If you are really worried about storage usage […]"*
- *"This could be very annoying but it is simply solved"*
- *"[…] This will make your experience a lot less frustrating"*

or in a criticism towards a technological issue, as in the following cases:

- *"This could work, but feels really awful"*
- *"This is extremely ugly for loop construction."*

As for comments to answers, we found they are very rich in affective lexicon too. We observe that Stack Overflow users express a wide variety of affective states in comments. We speculate that this might occur because comments are seen as a 'free zone' since reputation mechanisms do not apply to comments. More in detail, as for positive comments we found that the main affective states are gratitude, as in

- *"Thanks for the feedback, it was a pleasure!"*

wishes, as in

- *"Happy coding!"*

and positive feelings linked to satisfaction and happiness for the help provided, as in the following example

- Asker*: "Thanks, that helped. Case closed!"*- Answerer*: "Thanks for the feedback! It was a pleasure!"*

As for negative comments, we observe a wider variety of emotions. In some cases the negative polarity of lexicon actually conveys a negative attitude towards the interlocutor (in this case, the author of the question)

- *"Didn't notice the horrid inline jQuery"*
- *"Added some instructions for the really hopeless cases"*
- *"Arrrghhh, how I hate those people who downvote answers without leaving a comment as for why the downvote…"*

Conversely, negative lexicon may also be used to convey a positive attitude towards the reader. In such cases, the writer attempts to convey signal of empathy towards the interlocutor, as in the following examples, where people apologize for not being able to provide further help:

- *"To explain my regrettably unfriendly comment (sorry about that)."*
- *"You could try this one (not optimal, I am afraid)"*
- *"I'm afraid I can't help you any further with this issue!"*

Furthermore, users employ a negative lexicon also for expressing opinions on controversial technical issues, as in:

- Asker*: "But what if you do have to worry about spaces in your filenames?"* - Answerer*: "Then you've got major problems! Let me meditate on it; it is extremely unpleasant, whatever."*
- *"Sorry for all the editing but this is a ridiculously complex issue"*

3.4 Domain-dependence of Sentiment Analysis

Due to the presence of domain-specific lexicon, examples of false positives in negative sentiment detection emerge from the analysis of both comments and answers, such as in

- *"You are vulnerable to this bug"*
- *"What is the best way to kill a critical process"*
- *"I am missing a parenthesis. But where?"*

As already highlighted in Section 2, the domain-dependency of sentiment analysis tools is a known problem [15] meaning that applying a tool outside the domain in which it was tested, may produce unreliable results.

In the case of Stack Overflow, this bias is emphasized by the fact that a social Q&A site is explicitly designed for people looking for help because in trouble with a domain-specific issue. Therefore, discussion on Q&A sites are intrinsically skewed towards negative polarity because they are naturally rich in 'problem' lexicon, which does not necessarily indicate the intention to show any affective state, as in the following example, for which a negative score is inaccurately issued by SentiStrength:

- *"I have a trouble […]. Please, explain what is wrong"*

4. DISCUSSION AND FUTURE WORK

The findings of our qualitative investigation highlight open challenges for sentiment analysis in social software engineering and inspire directions for future work. First, in contrast with the current trend of using sentiment to operationalize the affect dimension in empirical studies, our analysis advocates in favor of the adoption of more appropriate affective state models. The wide variety of emotions, attitudes and opinions retrieved in our dataset confirm that affect is a quite complex phenomenon whose polarity is only one dimension of analysis.

Even when performed with state-of-the-art tools, measuring the polarity of a text is not sufficient to capture the attitude of the sender towards the recipient of a text, despite the use of affective loaded lexicon which may refer to technical issue rather than being addressed to the interlocutor.

Furthermore, the wide variety of affective states expressed in Stack Overflow posts recommends a more fine-grained investigation of the role of emotions. Depending on the specific goals addressed, researchers could be interested in issuing a discrete label describing the affective state expressed (e.g., frustration, anger, sadness, joy, satisfaction) as different affective states may be relevant to different context of interaction and tasks. For example, being able to identify harsh comments towards technical matters could be useful in detecting particularly challenging questions that have not been exhaustively answered, which is a goal addressed by current research on effective knowledge-sharing in Stack Overflow [2]. Conversely, detecting attitude towards the interlocutor could be of help for the community moderators that could intervene in order to guide the users' behavior towards a more constructive pattern of interaction towards cooperative problem solving. Indeed, previous research on success of questions has shown how strong negative emotions in follow-up discussions discourage participation [3]. In fact, even if Stack Overflow guidelines include a 'Be nice' section, in which users are invited to be patient and avoid offensive behavior, people might not be prepared for effectively dealing with the barriers of social media to non-verbal communication. This clearly emerges as an open problem in the Stack Exchange community as discussed by users, which complain about harsh comments mainly coming from expert contributors.

Finally, we underline the need for tuning state-of-the-art resources for sentiment detection. Indeed, an open challenge for sentiment analysis is to overcome the limitations induced by domain-dependent use of lexicon. This is consistent with Wittgenstein's *meaning-is-use* assumption [50], claiming that the meaning of an expression is fully determined by its use.

In our future work, we will improve lexicon-based sentiment analysis approaches and fine-tune state-of-the-art resources by exploiting semantic features [6] for appropriately dealing with domain-dependent use of lexicon, in order to distinguish accurately neutral sentences from emotionally loaded ones in Stack Overflow discussions.

5. ACKNOWLEDGMENTS

This work is partially funded by the ATS Romantic under the Apulian ICT Living Labs program and by the project 'Investigating the Role of Emotions in Online Question & Answer Sites', funded by MIUR (Ministero dell'Università e della Ricerca) under the program "Scientific Independence of young Researchers" (SIR).

6. REFERENCES

[1] Althoff, T., Danescu-Niculescu-Mizil, C., and Jurafsky, D. 2014. How to Ask for a Favor: A Case Study on the Success of Altruistic Requests. *In Proceedings of the 8th International AAAI Conference on Weblogs and Social Media (ICWSM 2014)*.

[2] Anderson, A., Huttenlocher, D., Kleinberg, J. and Leskovec. J. 2012. Discovering value from community activity on focused question answering sites: a case study of stack overflow. *Proc. of KDD '12*. ACM, 850-858.

[3] Asaduzzaman, M., Mashiyat, A.S., Roy, C.K., Schneider, K.A. 2013. Answering questions about unanswered questions of Stack Overflow, *Proc. of MSR 2013*, 97-100.B.

[4] Akkaya, C., Wiebe, J., Conrad, A., and Mihalcea, R. 2011. Improving the Impact of Subjectivity Word Sense Disambiguation on Contextual Opinion Analysis. *Proc. Of CoNNL '11*, 87-96.

[5] Austin, J 1962. *How to do Things with Words.* Oxford University Press, New York.

[6] Basile, P. and Novielli, N. 2015. UNIBA: Sentiment Analysis of English Tweets Combining Micro-blogging, Lexicon and Semantic Features. *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, ACL, 595—600.

[7] Batliner, A., Fisher, K., Huber, R., Spilker, J., No¨th, E. 2003. How to find trouble in communication. Speech Communication 40, 117–143.

[8] Brooks, M., Kuksenok, K., Torkildson, M.K., Perry, D., Robinson, J.J., Scott, T.J., Anicello, O., Zukowski, A., Harris, P., and Aragon, C.R. 2013. Statistical affect detection in collaborative chat. In *Proceedings of the 2013 conference on Computer supported cooperative work* (CSCW '13). ACM, New York, NY, USA, 317-328.

[9] Calefato, F., Lanubile, F., Marasciulo, M.C., Novielli, N. 2015. MSR Challenge: "Mining Successful Answers in Stack Overflow." In *Proc. 12th IEEE Working Conf. on Mining Software Repositories* (MSR '15), Florence, Italy, May 16-17, 2015.

[10] Clore, Gerald L.; Ortony, Andrew; Foss, Mark A. 1987. The psychological foundations of the affective lexicon. *Journal of Personality and Social Psychology*, Vol 53(4), Oct 1987, 751-766.

[11] Cowie, R. 2006. *Emotional life, terminological and conceptual clarifications*. Available at the HUMAINE Portal: http://emotion-research.net/deliverables/D3i final.pdf

[12] Ekman, P. 1999. *Basic Emotions. Handbook of Cognition and Emotion*, John Wiley & Sons Ltd.

[13] Esuli, A. and Sebastiani, F. 2006. Sentiwordnet: A publicly available lexical resource for opinion mining. *Proceedings of LREC*, volume 6, 417–422.

[14] Fellbaum C. WordNet: An Electronic Lexical Database. *Language, Speech, and Communication*. MIT Press, 1998.

[15] Gamon, M., Aue, A., Corston-Oliver, S. and Ringger, E. 2005. Pulse: Mining customer opinions from free text. *LNCS 3646*, 121-132.

[16] Guzman, E. and Bruegge, B. 2013. Towards emotional awareness in software development teams. *In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2013)*. ACM, New York, NY, USA, 671-674.

[17] Guzman, E., Azócar, D., and Li, Y. 2014. Sentiment analysis of commit comments in GitHub: an empirical study. *In Proc. of the 11th Working Conf. on Mining Software Repositories (MSR 2014)*. ACM, New York, NY, USA, 352-355.

[18] Hu, M. and Liu, B. 2004. Mining and Summarizing Customer Reviews. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 168–177.

[19] Kucuktunc, O., Cambazoglu, B.B., Weber, I., and Ferhatosmanoglu, H. 2012. A large-scale sentiment analysis for Yahoo! answers. *In Proceedings of the fifth ACM international conference on Web search and data mining (WSDM '12)*. ACM, New York, NY, USA, 633-642.

[20] Lazarus R S (1991) *Emotion and adaptation*. New York: Oxford University Press.

[21] Le Tallec, M., Antoine, J.-Y., Villaneau, J. and Duhaut, D. 2011. Affective interaction with a companion robot for hospitalized children: a linguistically based model for emotion detection. *Proc. of the 5th Language and Technology Conference (LTC2011)*.

[22] Litman, D. Forbes-Riley, K., Silliman, S. 2003. Towards emotion prediction in spoken tutoring dialogues. In: *Companion Proceedings of the Human Language Technology Conference: 3rd Meeting of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL)*, Kluwer, Amsterdam, pp. 52–54.

[23] Liu, H.,Lieberman, H. and Selker, T. 2003. A model of textual affect sensing using real-world knowledge. *Proceedings of the 8th international conference on Intelligent user interfaces*, ser. IUI

[24] Saif M. Mohammad and Peter D. Turney. 2010. Emotions Evoked by Common Words and Phrases: Using Mechanical Turk to Create an Emotion Lexicon. In *Proc. of the NAACL HLT 2010 Workshop CAAGET '10*, pages 26–34.

[25] Saif Mohammad, Svetlana Kiritchenko, and Xiaodan Zhu. 2013. Nrc-canada: Building the state-of-the art in sentiment analysis of tweets. *Proc. SemEval 2013*, 321–327.

[26] Murgia, A., Tourani, P., Adams, B., and Ortu, M. 2014. Do developers feel emotions? An exploratory analysis of emotions in software artifacts. *In Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014)*. ACM, New York, NY, USA, 262-271.

[27] Neviarouskaya, A., Prendinger, H. and Ishizuka, M. 2011. Affect analysis model: Novel rule-based approach to affect sensing from text. *Nat. Lang. Eng.*, vol. 17(1), 95–135.

[28] Nielsen, F.A. 2011. A New ANEW: Evaluation of a Word List for Sentiment Analysis in Microblogs. *MSM 2011*: 93-98.

[29] Novielli, N., Calefato, F., Lanubile, F. 2014. Towards discovering the role of emotions in stack overflow. In *Proc. of 6th Int'l Workshop on Social Software Engineering* (SSE '14). ACM, New York, NY, USA, 33-36

[30] Novielli, N., de Rosis, F. and Mazzotta, I. 2010. User Attitude Towards an Embodied Conversational Agent: Effects of the Interaction Mode. *Journal of Pragmatics*, 42(9), Elsevier Science, 2385-2397.

[31] Novielli N. and Strapparava, C. 2013. The Role of Affect Analysis in Dialogue Act Identification. *IEEE Transactions on Affective Computing*, 4:439–451.

[32] O'Connor, B, Balasubramanyan, R., Routledge, B., and Smith, N. 2010. From tweets to polls: Linking text sentiment to public opinion time series. In *Intl AAAI Conf. on Weblogs and Social Media (ICWSM)*, volume 11, pages 122–129.

[33] Pang, B. and Lee, L. 2008. Opinion Mining and Sentiment Analysis. *Foundations and trends in information retrieval*, 2(1-2):1–135.

[34] Pennebaker, J. and Francis, M. 2001. *Linguistic Inquiry and Word Count: LIWC*. Erlbaum Publishers.

[35] Picard, R. W. 2000. *Affective Computing*. MIT Press.

[36] Plutchick, R. 1984. Emotions: A general psychoevolutionary theory. In K.R. Scherer & P. Ekman (Eds) *Approaches to emotion. Hillsdale*, NJ; Lawrence Ealrbaum Associates.

[37] Polanyi, L. and Zaenen, A. 2006. Contextual Valence Shifters. In *Computing Attitude and Affect in Text: Theory and Applications*, The Information Retrieval Series, 1-10

[38] Riloff, E., Patwardhan, S., & Wiebe, J. 2006. Feature subsumption for opinion analysis. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 440-448.

[39] Rosenthal, R., Nakov, P., Kiritchenko, S., Mohammad, S.M., Ritter, A., and Stoyanov, V. 2015. SemEval-2015 Task 10: Sentiment Analysis in Twitter. *In Proceedings of SemEval 2015*.

[40] Russell, J A (2003) *Core affect and the psychological construction of emotion*.

[41] Storey, M-A. 2012. The evolution of the social programmer. In *Proc of the 9th IEEE Working Conf on Mining Software Repositories* (MSR '12). IEEE Press, Piscataway, NJ, USA, 140-140.

[42] Leif Singer, Fernando Figueira Filho, Brendan Cleary, Christoph Treude, Margaret-Anne Storey, and Kurt Schneider. 2013. Mutual assessment in the social programmer ecosystem: an empirical investigation of developer profile aggregators. In *Proceedings of the 2013 conference on Computer supported cooperative work* (CSCW '13). ACM, New York, NY, USA, 103-116.

[43] Storey, M-A., Singer, L., Cleary, B., Figueira Filho, F. and Zagalsky, A. 2014. The (R) Evolution of social media in software engineering. In *Proceedings of the on Future of Software Engineering* (FOSE '14). ACM, New York, NY, USA, 100-116. DOI=10.1145/2593882.2593887

[44] Strapparava, C., and Valitutti, A. 2004. WordNet-affect: an affective extension of WordNet. *In Proceedings of the 4th International Conference on Language Resources and Evaluation*, 1083-1086.

[45] Strapparava, C., Valitutti, A., & Stock, O. 2006. The affective weight of lexicon. *Proceedings of the Fifth International Conference on Language Resources and Evaluation, LREC 2006.*

[46] Taboada, M., Brooke, J., Tofiloski, M., Voll, K.and Stede, M. 2011. Lexicon-based methods for sentiment analysis. *Comput. Linguist.* 37, 2, 267-307.

[47] Thelwall, M. Buckley, K. and Paltoglou, G. 2012. Sentiment Strength Detection for the Social Web. *Journal of the American Society for Information Science and Technology*, 63(1):163-173.

[48] Vapnik, V. 1995. *The Nature of Statistical Learning Theory*. Springer-Verlag.

[49] Wilson, T., Wiebe, J. and Hoffman, P. 2005. Recognizing Contextual Polarity in Phrase-level Sentiment Analysis. *Proc. of HLT '05*, 347–354.

[50] Wittgenstein, L. 1965. *Philosophical Investigations*. The Macmillan Company, New York, NY, USA.

The Social Developer: Now, Then, and Tomorrow

Terhi Kilamo, Marko Leppänen, and Tommi Mikkonen
Tampere University of Technology
Tampere, Finland
{terhi.kilamo, marko.leppanen, tommi.mikkonen}@tut.fi

ABSTRACT

The practice of software engineering needs both individual commitment as well as social interaction. It has long been widely recognized that communication problems are a major factor in the delay and failure of software projects. However, the patterns of communication that can be associated with the different development paradigms have gained less attention. In this paper, we present some views to the evolution of social dimensions in the light of software engineering methodologies and associated tools. To study this, we have surveyed a number of software developers working in industry to reflect our views into the state-of-practice in software development companies and shed light to the impact of distributed and agile development has had on developer communication. Towards the end of the paper, we provide some ideas for future research and draw some final conclusions.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*Programming teams*; D.m [**Miscellaneous**]: Software psychology

Keywords

Software engineering, development methodologies, tool support, socio-technical congruence, social developer

1. INTRODUCTION

The history is full of examples of engineering efforts that manifest the importance of cooperation. The pyramids, the Colosseum, the Moai Statues of Easter Island, the Great Wall of China, the Inca City of Machu Picchu, and the Hoover Dam are all landmarks of engineering as well as human collaboration and coordination, although the latter part is seldom addressed afterwards. However without careful coordination and management, these achievements would, as many other greatest moments of mankind, have been impossible. It has even been hypothesized that the language itself has been developed because of hominin reliance on tools and their coordinated manufacture [18]. So, the 'C's; namely communication, coordination, cooperation and collaboration, build on each other, adding more complexity and mutual trust, ultimately ending up into effective collaboration and integration.

SSE'15, September 1, 2015, Bergamo, Italy
© 2015 ACM. 978-1-4503-3818-9/15/09...$15.00
http://dx.doi.org/10.1145/2804381.2804388

Similarly to the above examples, the practice of software engineering needs both individual commitment as well as social interaction among the participants. When comparing how good a job software engineers do as individuals to their work as a team, the answer is obvious: It has long been widely recognized that communication problems are a major factor in the delay and failure of software projects [4]. In fact, while the software development community is already struggling with communication issues, the emerging practice of global software engineering is raising even more challenges. Software work is undertaken at geographically separated locations across national boundaries in a coordinated fashion, involving both real time (synchronous) and asynchronous interaction [25]. This emphasizes the need for timely, precise and uniform forms of communication across the planet.

As software development takes place at the global scale, the necessary communication should take place at such a scale as well. On this note, the practice of pull-based development is moving from its origin, open source software, to distributed development in general [9]. A recent study shows that the pull-based approach with support for independent development through developer specific branching is used not only to integrate code, but to do code review and, to a large extent, discuss new features [8]. Furthermore, the so-called socio-technical congruence [10], first pointed out by Conway [3], has become an important field for recent research activities (e.g.[30]). There the relation between humans, organizations, and technical artifacts is investigated.

We believe that the new ways of working in software development, culminating in Continuous Integration, Continuous Delivery, and DevOps, will have a profound effect on the fashion social cooperation and collaboration takes place in software development. In this paper, we present some views to the evolution of social dimensions and their evolution in software development in the light of the evolution software engineering methodologies and associated tools. We focus on the interaction between the stakeholders of software development. In particular, we are interested in how closely developers communicate with the customer representatives and with other developers, and how this work is partitioned between face-to-face time, documentation, and tools such as version control systems. Moreover, we also envision tools that will be needed in the future to support development activities as well as interactions between stakeholders taking part in the process. To study this, we have surveyed a number of developers working in industry to reflect our views on the state-of-practice in software development.

The rest of this paper is structured as follows. In Section 2, we discuss the different software development approaches, and in Section 3, we address the evolution of tools that support the development. In Section 4, we provide an insight into our observations regarding the social dimension in the

development. In Section 5, we provide results from our survey regarding the fashion social interaction takes place in software development. In Section 6, we give an extended discussion regarding our views and findings as well as provide some directions for future work. Finally, in Section 7, we draw some conclusions.

2. EVOLUTION OF DEVELOPMENT PRACTICES

The first formal definition for the term software engineering was introduced in the 1968 NATO software engineering conference – the very same event where the term "software crisis" was coined by some attendees [19]. Besides introducing the idea of software engineering, the attendees of the conference agreed that the design concepts essential to maintainable systems are modularity, specification, and generality [17].

The above key concepts enabled development models, where modularity was used as a basis for project executing and task allocation. In particular, Royce's Waterfall model [24], intended to be a straw man for the sake of argument, is often interpreted as meaning a method for sequentially executing specification, design coding, and testing phases, accompanied with definitions how these phases are executed to create software that satisfies user requirements. These phases are used to decompose the system into modules, with various characteristics [21]. These modules then form the basis for task division – each module can be allocated to a particular developer. Once the tasks have been allocated, the developer can implement the modules allocated to her in relative isolation, independently from other developers. This process is illustrated in Figure 1.

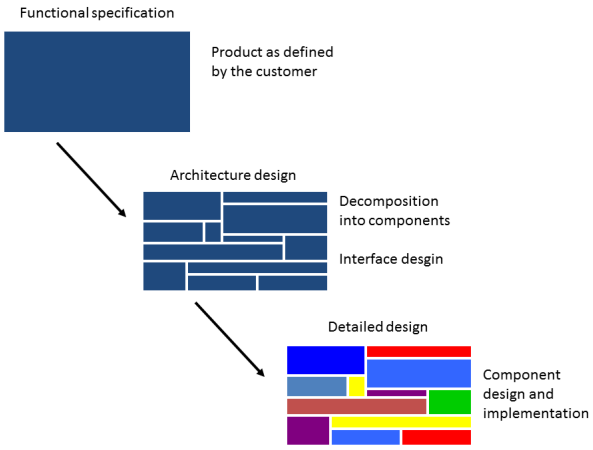

Figure 1: From requirements to architecture to components.

Although the modular structure of the software isolates the developers from each other, the isolation cannot be total. The modules interact with each other via interfaces and this interaction is reflected in the development team as communication. The implementation of interacting modules would be impossible without some degree of cooperation. Thus, the organization of the developers and their communication patterns resemble the structure of the software itself as originally noted by Conway's law [3]. Interestingly, the phenomenon Conway described, actually works the other way around; if the organization of the developers, either intentionally or unintentionally, promotes certain communication between the developers, a similar structure will be visible in the software.

Despite the known problems in the Waterfall approach – user requirements change, it is hard to get designs right on the first try, and the technologies evolve – this approach formed the baseline for software development for well over a decade. This was simply because it was considered as the best option to give structure and rationale in the development process [22].

The Waterfall model is reflected in various activities in the development practices. For example, approaches such as Capability Maturity Model (CMM) [23] can be considered as a derivative of the Waterfall model in its fundamental origins, although its later derivatives can be easily applied in various other settings. Similarly, using modularity for task allocation has led to several proposals for implementing modularity in programming languages, called modules, packages, classes, and so on. Combining modularity with generality has given us inheritance, where modularity is harnessed to support reuse without risking the possibility to use modules as the basis for task allocation. These have led to the perfection of the technical dimension of development, but they have not really assisted in supporting daily communication.

To overcome the challenges of the above mechanistic model, Agile approaches [15] advocate close and frequent communication between the client and the developers as well as among the developers themselves. While often implemented in the form of a development team sharing the same premises, encouraging frequent informal communication, distributed Agile development has also been proposed [29, 20]. In general, Agile software development, culminating in the Agile Manifesto[1], values "individuals and interactions over processes and tools", with the fundamental goal to embrace change that will be inevitably needed as development proceeds [1]. As an extension to Agile, lean software development introduced the concept of waste and proposed delivering value as soon as possible. This calls for several technical enablers, such as continuous integration [7] and smooth deployment [11]. In general, such reduction of time from completed implementation to deployment has resulted in DevOps [12] where developers and operators work as a team to deliver value to end users as quickly as possible.

3. SUPPORTING TOOLS AND OTHER INFRASTRUCTURE

Numerous tools and methods have been proposed to solve issues in different phases of projects, starting from capturing requirements and ending at customer documentation. These tools reflect how developers compose systems, and therefore they are closely related to the practices that are used.

The role of the customer in traditional software development models has been to provide information on what is needed. This is particularly visible in the Waterfall model, but applies also to some extent in the Agile setting. The customer's needs have traditionally been recorded in different systems and formats, be it requirements in a requirements document or a requirements management system, or

[1] http://agilemanifesto.org/

use cases in the product backlog in Agile development. For instance, Figure 2 presents a model where Scrum artifacts are depicted. Once the product backlog contains the requirements, it is the developers that usually create Sprint backlog, and later on implement items allocated in it.

Today, the above model is increasingly being challenged by involving users early on in the development. For instance, Koski et al. [14] report a development effort where a mission critical system – context where document-heavy development style is usually advocated – is built so that at least some of the end users are constantly collocated with the developers.

When considering the developer, the traditional two tools needed are the compiler and the debugger. These help the developers track down what takes place as programs are built and run. As long as the developers were working independently, there was little need for additional infrastructure. However as the number of developers working on the same system – and more recently, even on the same module – grew, facilities for managing the different versions became necessary. The same goes for configuration management systems; it was impossible to manage all configurations that were to be designed in the joint human memory of the development teams.

With increasingly complex configurations and frequent changes in software, constant, time-consuming compilations formed yet another impediment. To this end, build systems provide support for working with the same code base, which is compiled whenever the software is changed [6]. Distributed version control systems further allow development to be done in isolation and the integration of the work of the developers to be tooled. This push towards faster compilations and delivery to end user culminates in DevOps [12], where the developers are working similarly to and in collaboration with the operators, and deliver updates several times a day. This obviously requires a carefully designed interplay of numerous different development tools – and of the people who use them as well.

In addition to the tools that developers themselves use, there is a growing tendency to collect data from end users to support determining which direction the development should go to [5]. With such tools, it is possible to create controlled experiments to evaluate different options with actual end users, in particular in cases where it is next to impossible to reach a satisfactory, truly representative person to consider. The data collection setup resembles scientific experimentation. Moreover, collecting data obviously raises concerns with privacy, but at the same time, end user observation instead of direct communication with end users avoids several biases, such as non-response bias, recall bias, interviewer effects and types of response biases altogether. Thus, the analysis is more in the domain of inferential statistics, and not so much in communication. However, one must still follow the basic rules of scientific experimentation in order to avoid other kinds of subjectivity.

4. SOCIAL DIMENSIONS IN DEVELOPMENT AND DEPLOYMENT

Tools, methods and processes that are used to develop software also define the necessary interactions between the developers and the different stakeholders of the system. In the following, we address the three main eras of software development — Waterfall, Agile, and DevOps – based on the characteristic properties they pose. Obviously, the boundary between any of these approaches is a fine line, and therefore the fashion communication takes place should be considered as a continuum. For instance, executing the waterfall model several times in sequence results in an iterative development approach, which shares some of its characteristics with Agile development. Similarly, an Agile setup that is based on continuous integration and deployment obviously shares some of its properties with DevOps, while the latter also includes properties that are beyond what Agile approaches propose.

Waterfall era: Developers form teams that interact as they design the interaction between modules that are used for allocating tasks; the same paradigm is reflected in many programming languages, where module constructs are designed for such use. In contrast to direct interaction between developers, the interaction with users is vague. Requirements are gathered and documented in a separate document or into an information system, and they are only revised as the last resort. In contrast the deployment of the system takes a long time and requires deep collaboration between the developers and users of the system.

Agile era: Developers form tight-knit teams that share responsibility over the code the team has designed. Consequently, the interaction between developers is frequent, which is sometimes enforced with practices such as the Daily Scrum. As for interaction with the end users, there are different practices, but they share the idea that developers should focus on the actual development. For instance, in Scrum, the Product Owner manages the requirements in the product backlog, and negotiates with the developer team regarding how and in which order to implement them. Regarding deployment, the model is still fundamentally based on installations that require collaboration between developers and users.

DevOps era: Developers cooperate with system operators and end users to define actual needs. Once the needs are understood, the developers can single-handedly implement the changes across the whole system and deploy the outcome. Moreover, when no easy access to end users is available for eliciting requirements, analytic tools can be used to gather real usage data based on which decisions regarding the future of the software system can be made. As tools play a pivotal role in achieving this, they also take care of developer-to-developer communication regarding day-to-day issues, at least partially. However, regular meetings between developers can take place to synchronize tasks and to share common status. Thus, while often advocated as an Agile approach, many of the characteristics of DevOps are not really serving the ideals of the Agile Manifesto mentioned above, but are rather about tools and processes instead of interactions. Moreover, as features are deployed as soon as they are designed and implemented, fewer interactions are required.

To summarize, we claim that the social dimension in software engineering is experiencing a shift from developer-centric interaction to a setup where tools are largely automating this kind of communication. This shift can be seen in the emergence of pull-based communication and in how modern development environments support implicit coordination of work [2]. At the same time, the ability to deliver end-user value more rapidly implies that the developers must also interact with the users more directly. Developers

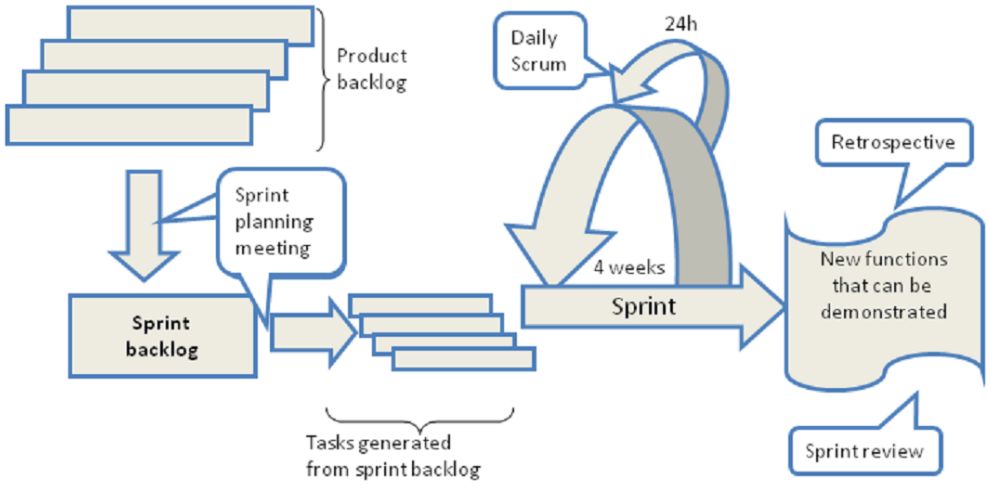

Figure 2: Artifacts and actions in Scrum.

can no longer only rely on mediators to deliver information regarding end-user needs. Figure 3 illustrates the communication levels of modern developers. At present, we lack the means to implement this in a way that is easy, casual, and practical.

5. SURVEY: STATE-OF-THE-PRACTICE IN INDUSTRY

In order to study the state-of-practice development setups in industry, we executed a personal opinion survey extended with some questionnaire-like open ended questions. The questions regarded the social dimensions in software development as well as practices that go along with the social part.

5.1 Study Setup

The role of the cross sectional survey [13] was to collect data on how software developers working in modern software intensive companies communicate with each other as well as with the customer, and what is the role of the development tools. The questions of the survey included background questions, where information was collected regarding the size of the team and the type of the project the developer is working with. In addition, we asked what the preferred communication mechanisms are.

Questions addressing the actual communication in the development team were presented in the following form:

- What feedback mechanisms you have in your project?
- What are the main sources of feedback in technical quality?
- What are the main sources of feedback for the validation of the features?
- What are the main communication ways while deciding the task allocation with the team?

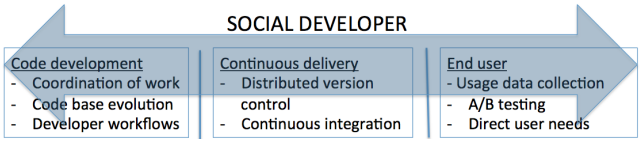

Figure 3: Illustration of the overarching dimensions of communication in modern software development.

with multiple choice questions. In addition to the mechanisms, we asked how well communication methods in the project suited the respondents (scale 1=not at all, 7=ideally), and with an open ended question, how would they improve communication within their projects.

The survey was designed by two researchers and first sent to a test group consisting of researchers working in the same research project with topics related to continuous software engineering and with knowledge of software development. Based on their feedback, the survey questions were improved. The survey was disseminated to respondents working in Finnish software intensive companies by email, by direct messaging and by face to face discussions. The companies were allowed to distribute the survey as they saw fit and forward it to their partners. Among the companies were 20 either large enterprises or SMEs. Additionally several small startups were contacted. The questions were in Finnish. One of the companies requested an English version, which was sent; however the results reported in this paper only cover responses from the Finnish survey, and extending the survey remains a part of future work.

5.2 Results

Background: Of the 25 responses to the Finnish survey, 13 worked in a customer project, 11 in product development, and 1 in a non-specified project type. In the open ended question, 8 respondents were working in a co-located team and 12 in a distributed team. The rest included 1 startup, 3 mixed environments with both co-located and distributed aspects, and 1 Free/Libre/Open source developer (Figure 4). The distributed teams were mostly distributed over different cities in Finland (total of 8 answers). Out of the 4 globally distributed teams, 2 mentioned the US as the other site, and one team had a developer in Costa Rica. One global team was not specified.

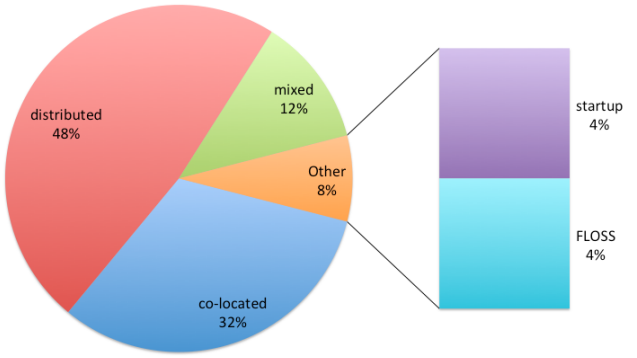

Figure 4: The distribution of the type of the respondents' project work based on their open ended question responses.

Regarding the best-suited way to communicate in the project, instant messaging was most popular (92%), closely followed by face to face contact among team members (84%). In addition, emails (64%) and task management systems (60%) rated rather high. In contrast, discussions with end users were mentioned only by 20% of the respondents. The detailed results are presented in Figure 5.

Available feedback mechanisms: The most commonly provided feedback mechanisms were version management systems (84%) and task management systems (76%) as shown in Figure 6. In addition, bug reporting systems and direct feedback from the clients were reported by 64% and 64% of the respondents, respectively. The least common feedback mechanism was testing team (32% of respondents), probably reflecting the fact that in DevOps type of development, there is little need for separate testing function.

Technical quality: The most important feedback system regarding technical quality was bug reporting system (used by 44% of the respondents), closely followed by direct feedback from customers (40%) and peer reviews (40%). The responses are presented in detail in Figure 7.

Feature validation: Regarding feature validation, design meetings were regarded as the most important mechanism (54%), followed by daily meetings (42%), project manager (42%), and – perhaps surprisingly – instant messaging systems (38%). Figure 8 presents all the responses.

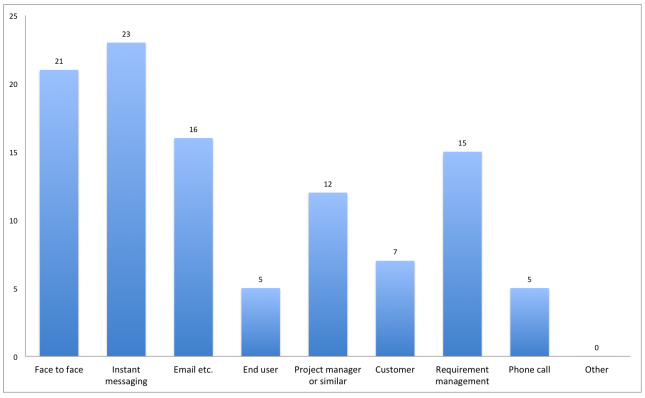

Figure 5: Preferred communication mechanisms.

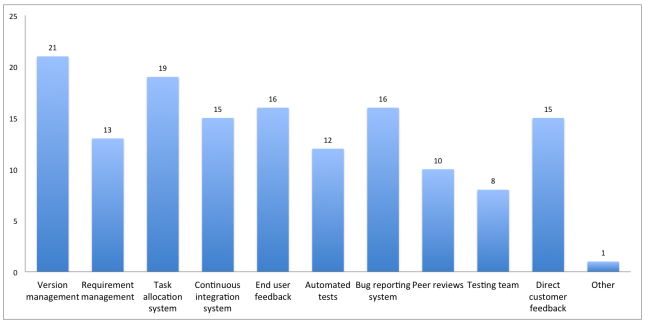

Figure 6: Available communication mechanisms.

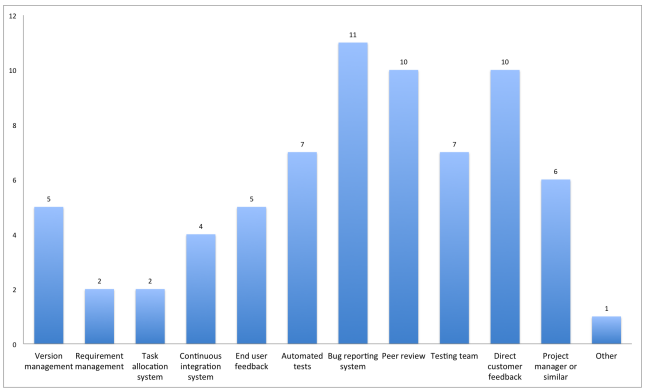

Figure 7: Technical quality.

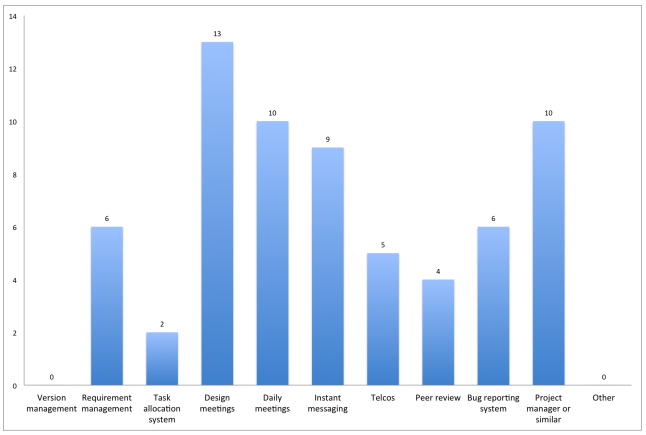

Figure 8: Feature validation.

Task allocation: Face-to-face meetings are the most important mechanism for task allocation, advocated by 76% of the respondents. In addition, project manager (60%) and task management systems (44%) play an important role. All the responses are presented in detail in Figure 9.

Improvements: The main improvements that were proposed were still to have the team co-located in order to improve communication between developers. Moreover, tighter integration of the customer in the project was proposed by several respondents. In addition, some of the respondents were critizing informal discussions with product owners and customers in particular over the phone, where it is difficult to record the requirements right.

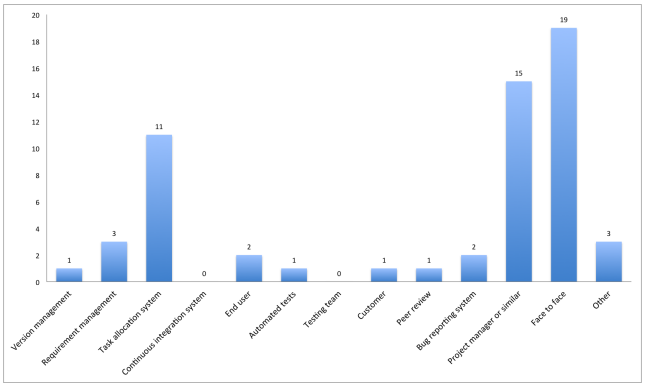

Figure 9: Task allocation.

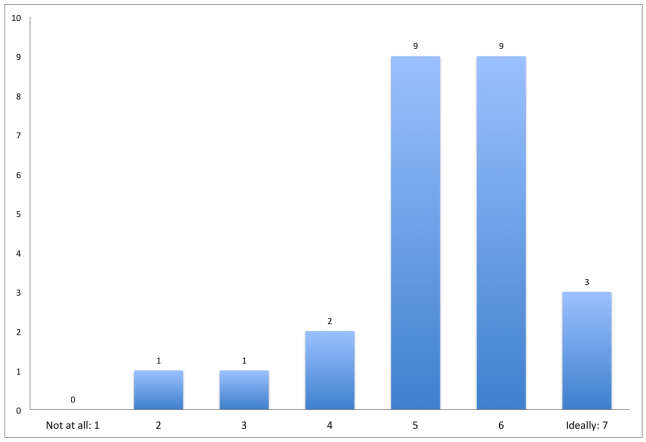

Figure 10: Developer satisfaction with the suitability of the communication channels used.

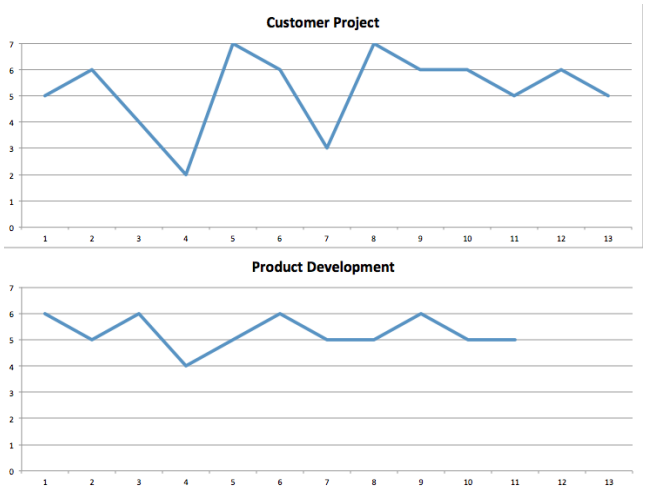

Figure 11: Difference between communication satisfaction of product development and customer projects.

5.3 Summary and Discussion

Based on the data, the need for non-disruptive and asynchronous communication is evident, and also its rationale can be easily explained. Developers simply appreciate fast communication in a tool assisted fashion. In contrast, interruptions that break the development flow are annoying, and for example instant messaging is preferred to phone calls – likely due to the less disruptive nature of messaging, leading in fewer context switches.

Making things visible through tools, boards, war rooms, tickets etc. is highly appreciated. The need for communication tools and open communication instead of meetings and calls came through. While few developers wanted direct interaction with the client, it was also mentioned as something to be improved. Consequently there is room for better feedback loops from the users to the developers, preferably through asynchronous means. While peer reviews are not among the most used feedback channels, they were mentioned as something to do more in order to foster cooperation. Finally, while commonly available, end user feedback appears to be a deprecated way to interact, at least among developers.

Mostly the developers were satisfied with the used communication methods as seen in Figure 10. However, the developers working in product development tended to be more satisfied than those working on customer projects (Figure 11. At the same time, the most satisfied developers were also among those working in customer projects. This may be caused by the customer influence on communication.

5.4 Limitations

Internal validity: There is an unavoidable selection bias in the selection of the companies due to the research project context and the local IT sector. This is mitigated by the wide range of companies contacted. There is also an evident self-selection bias as the companies distributed the survey internally as they saw fit and thus may have chosen certain types of development teams and products as their focus. Additionally, the respondents may have chosen to answer based on their own interests in DevOps and modern tool and communication chains.The wording of the questions can further mean different things to different people. Some are also more likely to use the extremities of the answer scale than others.

External validity: The main threat to the external validity is the companies being only Finnish and mainly operating in the Tampere region in Finland. This may lead to the results be less generalizable. This threat is mitigated by the amount of geographically distributed teams, some of which are also globally distributed.

6. DISCUSSION

The practice of software engineering seems to be driven with the necessity to do software that is just about as complex as we can implement with the present tools, methods and practices. Thus, the evolution in software engineering practices has had a profound effect on how developers interact with customers, end users, and each other. The mechanistic model visible in the Waterfall model has been replaced with approaches that build on interaction and rapid implementation. Moreover, the recent trends are leading towards interactions that manifest in information systems that help in the development and deployment of the resulting software. The faster deployment in turn supports interactions

by providing data regarding the actual use of the system as well as user feedback.

The need for interaction between developers and with other stakeholders of software projects has already changed the characteristics of a successful software engineer. Today, a representative software engineer is almost an opposite of the traditional caricature nerd – excellent people skills are needed to succeed in customer-related operations and in in-house development, where the developers work closely together. This has placed totally new requirements for the education, as each developer must be able to solve problems individually but also be a team player. This also indicates the need for asynchronous communication despite the increased focus on team cohesion.This was also affirmed by our study, as the developers seemed to value instant messaging over meetings.

In addition to the increasing amount of interaction, tool chains that allow faster delivery of end-user value are gaining importance. Based on interactions with various stakeholders, the developers are introducing new features that the customers or end users need, with tools that allow hundreds or even thousands of changes per day. Then, understanding what is truly valuable either is evaluated with further interactions, or by using additional tools, to the extent that at times the development is simply a series of changes that need no direction at all [5]. Moreover, while it is possible to operate in a mode where close interactions between developers and other stakeholders result in new features, comprehensive updates and refactoring may require a change in the mode of operation in the development organization [26].

As individual developers are able to carry out changes in the system single-handedly and in isolation, communication among developers as well as between the developers and customers requires using information systems. For the former, it is common to use wikis, version control systems and instant messaging systems to communicate progress in the development. In contrast, when communicating with the customer, requirements management systems such as Jira[2] are used. According to our study, while developers seem to appreciate face to face interaction within the team, asynchronous communication means are generally valued.

Since none of the communication systems used is intended to be used as a communication system among different parties, we expect that the increasing social interactions between developers and other stakeholders will result in new type of information systems, that resemble both traditional requirements management systems as well as social media. This trend is already visible in the developer's use of social media [28, 27]. Today, these new properties could probably best be studied by using mashups that connect data from various sources (e.g. [31, 16]) instead of focusing on each aspect in isolation. There is an intricate interplay of social dimension from writing code in isolation to communication of end user needs that requires further research focus. Obviously there are additional new, unforeseen directions for future research – new socio-technical implications, coordination patterns, the interplay of development in isolation and in collaboration and stakeholder connections.

Understanding how socio-technical congruence will manifest itself in the new ways of working will further provide excellent research topics, which will also help understanding how software of the future should be created and maintained. Moreover, there is a lot of room for tool infrastructure that helps in maintaining the congruence as well as supports the different types of interactions.

[2]https://www.atlassian.com/software/jira

7. CONCLUSIONS

The liberation of the development team to operate on their own terms, rather than being forced to a joint process, has advanced in parallel with the introduction of tools that truly help the developer in creating the software. This evolution is leading to shared responsibility over the code base and, to increasing extent, also over satisfying end-user expectations. Moreover, the characteristics that a successful developer must display are rapidly evolving as a part of this change as well.

In general, the transition from allocating responsibility in accordance with software architecture to a model where developers bear a shared responsibility will have consequences that can only be hypothesized at this time. The burning open questions include what will be the socio-technological congruence of future software systems, how should this be taken into account in their design and development infrastructure, and what consequences this will have on the actual development. However, it is apparent that a modern software developer does not shy away from asynchronous communication and development tools as communication media.

8. ACKNOWLEDGMENTS

The authors wish to thank Digile's Need4Speed program[3] funded by the Finnish Funding Agency for Innovation Tekes[4] for its support for this research.

9. REFERENCES

[1] K. Beck. *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2000.

[2] K. Blincoe and D. Damian. Implicit coordination: A case study of the rails oss project. In *Open Source Systems: Adoption and Impact*, pages 35–44. Springer, 2015.

[3] M. E. Conway. How do committees invent. *Datamation*, 14(4):28–31, 1968.

[4] B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(11):1268–1287, 1988.

[5] D. G. Feitelson, E. Frachtenberg, and K. L. Beck. Development and deployment at Facebook. *IEEE Internet Computing*, 17(4):8–17, 2013.

[6] M. Fowler. Continuous Integration, May 2006.

[7] M. Fowler and M. Foemmel. Continuous integration. *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf*, 2006.

[8] G. Gousios, Z. Andy, and S. Margaret-Anne. Work practices and challenges in pull-based development: The integrator's perspectivel. In *Proceedings of the 37th International Conference on Software Engineering*, pages 358–368. ACM, 2015.

[9] G. Gousios, M. Pinzger, and A. v. Deursen. An exploratory study of the pull-based software

[3]http://www.n4s.fi/

[4]http://www.tekes.fi/en/tekes/

development model. In *Proceedings of the 36th International Conference on Software Engineering*, pages 345–355. ACM, 2014.

[10] J. D. Herbsleb. Global software engineering: The future of socio-technical coordination. In *2007 Future of Software Engineering*, pages 188–198. IEEE Computer Society, 2007.

[11] J. Humble and D. Farley. *Continuous delivery: Reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

[12] J. Humble and J. Molesky. Why enterprises must adopt devops to enable continuous delivery. *Cutter IT Journal*, 24(8):6, 2011.

[13] B. A. Kitchenham and S. L. Pfleeger. Personal opinion surveys. In *Guide to Advanced Empirical Software Engineering*, pages 63–92. Springer, 2008.

[14] A. Koski and T. Mikkonen. Rolling out a mission critical system in an agilish way: Reflections on building a large-scale dependable information system for public sector. In *RCoSE'15*, 2015.

[15] R. C. Martin. *Agile software development: principles, patterns, and practices*. Prentice Hall PTR, 2003.

[16] A.-L. Mattila, T. Lehtonen, K. Systä, H. Terho, and T. Mikkonen. Mashing up software management, development, and usage data. In *RCoSE'15*, 2015.

[17] M. D. McIlroy, J. Buxton, P. Naur, and B. Randell. Mass-produced software components. In *Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen, Germany*, pages 88–98. sn, 1968.

[18] T. Morgan, N. Uomini, L. Rendell, L. Chouinard-Thuly, S. Street, H. Lewis, C. Cross, C. Evans, R. Kearney, I. de la Torre, et al. Experimental evidence for the co-evolution of hominin tool-making teaching and language. *Nature communications*, 6, 2015.

[19] P. Naur and B. Randell. Software engineering: Report of a conference sponsored by the nato science committee, garmisch, germany, 7-11 oct. 1968, brussels, scientific affairs division, nato. 1969.

[20] M. Paasivaara, S. Durasiewicz, and C. Lassenius. Using scrum in distributed agile development: A multiple case study. In *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on*, pages 195–204. IEEE, 2009.

[21] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[22] D. L. Parnas and P. C. Clements. A rational design process: How and why to fake it. *Software Engineering, IEEE Transactions on*, (2):251–257, 1986.

[23] M. Paulk. *Capability maturity model for software*. Wiley Online Library, 1993.

[24] W. W. Royce. Managing the development of large software systems. In *proceedings of IEEE WESCON*, volume 26. Los Angeles, 1970.

[25] S. Sahay. Global software alliances: the challenge of'standardization'. *Scandinavian Journal of Information Systems*, 15(1):11, 2003.

[26] J. Savolainen, N. Niu, T. Mikkonen, and T. Fogdal. Long-term product line sustainability with planned staged investments. *Software, IEEE*, 30(6):63–69, 2013.

[27] M. Squire. Should we move to stack overflow? measuring the utility of social media for developer support. In *Proceedings of the 37th International Conference on Software Engineering*, pages 219–228. ACM, 2015.

[28] M.-A. Storey, L. Singer, B. Cleary, F. Figueira Filho, and A. Zagalsky. The (r) evolution of social media in software engineering. In *Proceedings of the on Future of Software Engineering*, pages 100–116. ACM, 2014.

[29] K. Sureshchandra and J. Shrinivasavadhani. Adopting agile in distributed development. In *Global Software Engineering, 2008. ICGSE 2008. IEEE International Conference on*, pages 217–221. IEEE, 2008.

[30] M. Syeed. *On the Socio-Technical Dependencies in Free/Libre/Open Source Software Projects*. Publication 1300, Tampere University of Technology, Tampere, Finland, 2015.

[31] J. Yu, B. Benatallah, F. Casati, and F. Daniel. Understanding mashup development. *Internet Computing, IEEE*, 12(5):44–52, 2008.

Preparing Next Generation of Software Engineers for Future Societal Challenges and Opportunities

Gordana Dodig-Crnkovic
Chalmers University of Technology
& University of Gothenburg, Sweden
gordana.dodig-crnkovic@chalmers.se

ABSTRACT

As a global community we are facing number of existential challenges like global warming, deficit of basic commodities, environmental degradation and other threats to life on earth, as well as possible unintended consequences of AI, nano-technology, biotechnology, and similar. Among world-wide responses to those challenges the framework programme for European research and technological development, Horizon 2020, have formulated the *Science with and for Society Work Programme*, based on *Responsible Research and Innovation* with a goal to support research contributing to the progress of humanity and preventing catastrophic events and their consequences. This goal may only be reached if we educate responsible researchers and engineers with both deep technical knowledge and broad disciplinary and social competence. From the perspective of experiences at two Swedish Universities, this paper argues for the benefits of teaching professional ethics and sustainable development to engineering students.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**] Computer and Information Science Education

General Terms

Theory.

Keywords

Social Software Engineering, Responsible Research and Innovation, Engineering Ethics, Research Ethics, Computer Ethics, Sustainable Development, T-shaped Engineer.

1. INTRODUCTION

"The global community is facing Grand Challenges. The European Knowledge Society must tackle these through the best analysis, powerful actions and increased resources. Challenges must turn into sustainable solutions in areas such as global warming, tightening supplies of energy, water and food, ageing societies, public health, pandemics and security. It must tackle the overarching challenge." The Lund Declaration, 2009 [1]

Our time is often characterized as the *age of complexity*. Increasing capability to deal with complexity is a direct consequence of the development of computing machinery and the ability to program it. Ubiquitous use of information processing (computational) devices that are programmed to control, predict or simulate variety of processes enables us to face complexity and better understand and anticipate the behaviors of complex systems. There is however a significant difference between calculating offline a mathematical function "in abstracto" and controlling a paper mill in practice. The difference is in level of complexity, context dependence and social embeddedness of technological solutions for practical problems.

However, education of new generations of software engineers often focus on training abstract skills without careful consideration of the role of embeddedness of technology into context. It shows considerable inertia when it comes to introducing new views of computing and in particular new understanding of how in the best way software engineer can prepare for his/her future professional work in the real-world context of application and its different aspects, especially social side of it. Recently this aspect of research and innovation attracted considerable attention, through the notion of *Responsible Research and Innovation (RRI)*, [2-4], defined as:

"*a transparent, interactive process by which societal actors and innovators become mutually responsive to each other with a view to the (ethical) acceptability, sustainability and societal desirability of the innovation process and its marketable products (in order to allow a proper embedding of scientific and technological advances in our society).***"** Von Schomberg [2]

This requirement and expectation of responsible research and development is connected to the concept of the *Triple Helix,* where universities stand for knowledge that is applied in industry in accordance with societal needs. The university-industry-government mutually interconnected relationships in the form of triple helix are described in Leydesdorff [5], and Ranga and Etzkowitz [6], that capture the transition from an industry-government relation characteristic for the industrial society to an increasingly tripartite relationship between industry-government-university in the emerging data-information-knowledge society.

The expectations are increasing for integrative, responsible research and innovation, aligned with societal needs through challenge-driven research and innovation, where scientific progress is connected to the sustainable development. The novelty in this approach is that societal involvement is present during the whole process, which is particularly important for the ICT,

SSE'15, September 1, 2015, Bergamo, Italy
ACM. 978-1-4503-3818-9/15/09...$15.00
http://dx.doi.org/10.1145/2804381.2804389

biotechnology, robotic, nanotechnology and similar emerging fields.

Societal challenges for the framework for European research and technological development Horizon 2020 are formulated in the *Science with and for Society Work Programme*, which is meant to "help build effective cooperation between science and society, to recruit new talent for science and *to pair scientific excellence with social awareness and responsibility*" (Science with and for Society Work Programme 2014-2015). This new approach encourages all stakeholders (involved citizens, researchers, business, policy makers, etc.) to interact throughout the research and innovation process and to coordinate and align both the process and its outcomes with societal values and needs, in accordance with Responsible Research and Innovation (RRI).

Process includes identification and prioritizing of research and innovation goals, accountability, values and transparency, where products are in accordance with societal values and needs (such as sustainability, safety, privacy, equity, diversity, etc.)

2. ORGANIZATIONAL ADAPTATION IN THE ERA OF COMPLEXITY AND CONTINUOUS CHANGE

As a consequence, on a systemic level there is a necessity of defining *social/organizational responsibility* in addition to customary *personal responsibility* [7]. We should take into account both intended and unintended consequences of research and technology in a preferably *anticipatory and learning process* that will in the first place prevent incidents and accidents and in the worst case *mitigate* their consequences, [8-13]. Present day dominantly *reactive policies* must be replaced by *anticipatory practice* based on scenario analyses, and foresight grounded in experiences (social learning) collected on a systemic level [7].

The dominant underlying structure of contemporary global society is its organisation in networks of networks of interacting agents. Each individual belongs to a variety of networks, which define their different roles as stakeholders in various aspects of research and technology. In this context complexity and trans-disciplinarity /inter-disciplinarity comes as an important aspect of research and development.

This new triple-helix type of relationship between research, society and industry/business necessitates *shared* ability of a broader view – socially aware researchers, engineers, citizens, politicians and businessmen. Interdisciplinary literacy and inclusion of "soft" aspects that promote pro-social value systems are becoming necessary.

To start with, the new understanding of the role of research and innovation as defined in Responsible Research and Innovation (RRI), Von Schomberg [2,3] must permeate education in the first (Bachelors), second (Masters) and third (PhD) cycle programmes. One of important components of the development of RRI is to educate engineers with shared professional goals and value systems.

3. THE UNIQUENESS DEBATE AND ENGINEERS TEACHING ETHICS

Even though ethics has an essential socio-cultural dimension the *basic principles* of ethics are constant, no matter in which area they might be applied. Thus the *fundamental principles* of medical ethics, legal ethics, and computer ethics are not different from one another. However, new situations related to the computers do raise new questions about *how* these principles are to be applied, phenomenon defined by Moor [14] as *policy vacuum* (that is the lack of previous experiences of similar situation and absence of best practices and policies) as elaborated in Johnson [15], Tavani [16] and Barger [17]. The important question was rised whether ethics of computing/ computer science/ software engineering is *unique* or simply an application of general ethical principles, known as the *uniqueness debate*. The notion of *uniqueness* is based on the comparison. Similarity and difference are *relative concepts*. *Uniqueness* is a matter of focus and context. Looking at the set of all possible ethical problems, different patterns can be recognized permitting their grouping into medical ethics, political ethics, environmental ethics, business ethics etc. The criteria for grouping problems within certain fields are several, and one of them is the *importance* of the ethical problem. The other is its specific and *unique* character. Moreover, Tavani [16] argues that the computer ethics issues are both philosophically interesting and deserving our attention, no matter whether those issues are unique ethical issues.

As a consequence of the uniqueness debate predominant understanding have emerged that ethics of computing/ computer ethics/ engineering ethics are unique ethics fields founded and lead by engineers and computer scientists with interests in ethics (such as J. Moore, H. Tavani, K. Miller and many others) and the opinion prevailed that it would be appropriate for engineers and researchers to teach courses in the subject.

4. EXPERIENCES FROM PROFESSIONAL ETHICS COURSES AT MÄLARDALEN UNIVERSITY

In the year 2003 I have developed the first course in Professional Ethics at Mälardalen University, intended primarily for Science and Engineering students, especially within CS, SE and Robotics. Its outline was based on equivalent courses at other universities worldwide. The course has been taught continuously since then for the first, second and third cycle programmes. The emphasis is on *cultivating ethical sensibility*, the development of *moral autonomy*, *ethical pluralism* and *critical thinking*.

Course includes lectures, with guest lectures given by researchers or practitioners with relevant experiences, individual presentations of students on the approved topic of interest, class discussions, and mini-conference where research papers are presented. Students are expected to write class notes containing their own reflections in connection to each lecture. Results of experiences with those courses have been published in a number of publications; where [8] is presenting social, ethical, and professional issues of computing curricula, [9] argues for the importance of teaching professional ethics to computer science students, [10] discusses professional ethics in software engineering curricula, while [11] focuses on professional ethics in computing and intelligent systems. References [19]-[25] are articles written in collaboration with my students who participated in Professional Ethics courses.

At present, course is given as a PhD course. It provides an insight into the ethical problems important for professionals in Engineering and Science. It forms a framework in which professional and ethical issues can be analyzed, and builds up an awareness of various views of ethical issues as well as the ethical responsibilities of professionals. The topics include, among others, the social context of a profession, conflicts between loyalties to different principles such as safety and economy, precautionary principle and environmental impact, integrity,

privacy, ownership, etc. Fundamental moral theories are presented as the introductory part of the course.

We discuss Codes of Ethics (such as IEEE, ACM, Responsible Conduct of Research, SE Code of Ethics), and examine a series of case studies, which have led to ethical problems. At the same time we develop critical thinking and argumentation techniques.

Number of articles produced in the course since 2003 have been published at international conferences, journals and as chapters in Licentiate and PhD theses.

In sum, our experiences of the course have been very positive. Students have participated actively in discussions, case studies and research on chosen topics. Even predominantly technically minded students were able to assimilate and use ethical concepts and develop ethical argument. The examination forms for the course are the writing of a research paper on an ethical topic of interest and an oral presentation of a chosen topic (such as safety, security, intellectual property, environmental ethics, privacy and personal integrity, etc.) followed by an in-class discussion led by the students responsible for the actual presentation. Course evaluation results show that students experienced the course as useful and relevant for their future professional activities.

Three industrial PhD students have included their articles written in the course as specific chapters on ethical aspects of their thesis work: Larsson M. [23], Larsson S. [24], and Kade [25]. One more Phd Student, Jägemar [22] is planning to do so. They have investigated ethical consequences of software testing practices, effectiveness of the software development teams related to different ethical attitudes, ethical aspects of motion capture and ethical and cognitive sustainability aspects of mobile devices. A number of other students have published their articles written in the course on professional ethics in international journals and at CEPE and E-CAP conferences.

A more detailed description of professional and ethical issues of software engineering curriculum applied in Swedish academic context of Mälardalen University can be found in [12]. Apart from the courses in professional ethics we also included Professional Ethics lecture as a part of courses in Research Methodology and Computing and Philosophy [13]. In hindsight, we can argue that ethics courses provide considerably better and more long lasting effect in teaching social aspects of engineering, than isolated lectures on ethics in other courses, which inevitably bring a lot of repetition and never reach necessary depth and scope.

5. COURSES IN ETHICS AND SUSTAINABLE DEVELOPMENT AT CHALMERS UNIVERSITY OF TECHNOLOGY

Interesting to mention, at the Software Engineering in Society (SEIS) track of the recent ICSE 2015 conference question was discussed about how many institutions, that participants were affiliated with, have courses teaching social aspects of software engineering, and specifically how many ethics courses. Surprisingly, there were very few such courses. If the vision of RRI is to become reality, there should be adequate education in social aspects of software engineering in all cycles of education.

The Swedish Higher Education Authority requirements that are defined in the Higher Education Act, include three qualitative targets for all courses and programmes: *knowledge and understanding, competence and skills and judgment and approach.* Courses in social aspects such as courses in ethics and sustainable development are definitely contributing to the category of *judgment and approach* with their explicit focus on values and broader societal context of research and engineering.

Some of the important questions when including social (and in particular ethical) aspects in engineering education are: Should ethics be a part of other courses (one lecture in every course, like Ethics and Cyber-Physical Systems) or should it be taught as a separate course? Would it be better if professional ethicists and philosophers (or environmental professionals in case of sustainability) would be engaged to teach courses like professional ethics or sustainable development, respectively, instead of professionals in computing? I would like to argue, based on the outcome of the *Uniqueness debate*, and formulations in *Social and Professional Issues* in Computing Curricula 2103 [26], that we engineers and researchers can and should be teaching social aspects of engineering, including ethics and sustainable development, to our students. It requires an initial effort, but experiences show that it is possible to do appropriately.

The development and implementation of the third cycle course Research Ethics and Sustainable Development at Chalmers University of Technology can be instructive in this context. The course is compulsory for all PhD students at Chalmers and consists of lectures with in-class discussions, guest lectures and group work, including peer review of individual essays that present each participant's thesis research project from the perspective of research ethics and sustainable development. As a framework as a basis for discussions we provide ethical review protocol with a list of possible ethical concerns and instructions on stakeholder analysis. Essays are discussed in groups and presented at a mini-conference at the end of the course, both individually (in a form of lightening talk) and then also in a group, which analyses individual essays on a more general level, searching for commonalities and differences, patterns and possibilities for improvements. Essays are written so that they can be published as a part of a PhD thesis. Present course that is centered around individual PhD research projects is a result of the development of a previous version of the course where group work consisted in discussion of specific general topics such as diversity, equity, publishing ethics, etc. Our experiences indicate that students show much more interest and engagement when starting from their own work and then critically think and relate to the research of their colleagues then when discussing in abstracto.

6. CONCLUSIONS

"With respect to system thinking, a T-shaped person is one who has technical depth in at least one aspect of the system's content, and a workable level of understanding of a fair number of the other system aspects. Many pure computer science graduates are strongly I-shaped, with a great deal of depth in software technology, but little understanding of the other disciplines involved in such areas as business, medicine, transportation, or Internet of Things. This leaves them poorly prepared to participate in the increasing numbers of projects involving multi-discipline system thinking." [18]

Boehm and Koolmanojwong Mobasser [18] address the necessity of the system thinking and educating of T-Shaped software engineers, which includes social aspects of software engineering.

The Science with and for Society Work Programme, with the requirement of Responsible Research and Innovation are supporting research process and products of research that will contribute to the advancement of humanity and avert catastrophic events or in the worst case mitigate their consequences. They

necessitate education of engineers with developed sensitivity to social aspects of engineering, including courses on research and engineering ethics and sustainable development.

This paper offers positive experiences from two ethics courses given at two different Swedish universities – Mälardalen University and Chalmers University of Technology, developed and taught by the author.

7. ACKNOWLEDGMENTS

The author wants to thank the three anonymous reviewers for their constructive comments.

8. REFERENCES

[1] The Lund Declaration, 8 July 2009. http://www.vr.se/download/18.7dac901212646d84fd38000336/

[2] Von Schomberg, R. 2011. Towards Responsible Research and Innovation in the Information and Communication Technologies and Security Technologies Fields (November 13, 2011). http://dx.doi.org/10.2139/ssrn.2436399

[3] Von Schomberg, R. 2013. A Vision of Responsible Research and Innovation, in Responsible Innovation, First Edition. Eds. Richard Owen, John Bessant and Maggy Heintz. John Wiley & Sons, Ltd. A constitution for Europe.

[4] Owen, R., Macnaghten, P.M., Stilgoe, J. 2012. Responsible Research and Innovation: from Science in Society to Science for Society, with Society. *Science and Public Policy 39* (6): 751/760.

[5] Leydesdorff, L. 1996. Emergence of a Triple Helix of University-Industry-Government Relations, Science and Public Policy, 23, pp. 279-86.

[6] Ranga, M. and Etzkowitz, H. 2013. Triple Helix Systems: An Analytical Framework for Innovation Policy and Practice in the *Knowledge Society, Industry and Higher Education* 27 (4): 237-262.

[7] Dodig-Crnkovic, G. and Çürüklü, B. 2011. Robots - Ethical by Design, *Ethics and Information Technology*, 14(1), 61-71.

[8] Dodig-Crnkovic, G. 2003. Computing Curricula: Social, Ethical, and Professional Issues. In *Proc. Conf. for the Promotion of Research in IT at New Universities and at University Colleges in Sweden.*

[9] Dodig-Crnkovic, G. 2006. On the Importance of Teaching Professional Ethics to Computer Science Students, Computing and Philosophy Conference, E-CAP 2004, Pavia, Italy. In: *Computing and Philosophy*, L. Magnani, Ed. Associated International Academic Publishers, Pavia.

[10] Dodig-Crnkovic, G. and Crnkovic, I. 2005. Professional Ethics in Software Engineering Curricula. Cross-disciplinarity in Engineering Education, CeTUSS.

[11] Dodig-Crnkovic, G. 2006. Professional Ethics in Computing and Intelligent Systems, In: *Proceedings of the Ninth Scandinavian Conference on Artificial Intelligence* (SCAI 2006), Espoo, Finland, Oct. 25-27.

[12] Dodig-Crnkovic, G. and Feldt, R. 2009. Professional and Ethical Issues of Software Engineering Curriculum Applied in Swedish Academic Context, HAoSE 2009 First Workshop on Human Aspects of Software Engineering, Orlando, Florida, October 25- 26.

[13] Dodig-Crnkovic, G. 2008. Computing and Philosophy Global Course, In: *APA Newsletter on Philosophy and Computers*, 08 (1).

[14] Moor, J. 1985.What is Computer Ethics, *Metaphilosophy* 16(4): 266-75.

[15] Johnson, D. G. 2003. Computer Ethics. In: *The Blackwell Guide to the Philosophy of Computing and Information* (Blackwell Philosophy Guides), 65-75.

[16] Tavani, H. 2002. The uniqueness debate in computer ethics: What exactly is at issue, and why does it matter? *Ethics and Information Technology* 4: 37–54.

[17] Barger, R.N. 2001. Is Computer Ethics unique in relation to other fields of Ethics? http://www.nd.edu/~rbarger/ce-unique.html

[18] Boehm, B. and Koolmanojwong Mobasser, S. 2015. System Thinking: Educating T-Shaped Software Engineers. IEEE 28th Conference on SE Education and Training CSEE&T In: *Proc. Int. Conf. on Software Engineering (ICSE) 2015*. Florence. Italy.

[19] Thekkilakattil, A. and Dodig-Crnkovic, G. 2015. Ethics Aspects of Embedded and Cyber-Physical Systems In: *IEEE Proceedings of COMPSAC 2015: The 39th Annual International Computers, Software & Applications Conference, Symposium on Embedded & Cyber-Physical Environments (ECPE)*. Taichung, Taiwan. July 1-5.

[20] Çürüklü, B., Dodig-Crnkovic, G., Akan, B. 2010. Towards Industrial Robots with Human Like Moral Responsibilities, In: *Proc. 5th ACM/IEEE International Conference on Human-Robot Interaction*, Osaka, Japan.

[21] Dodig-Crnkovic, G. and Horniak, V. 2008. Ethics and Privacy of Communications in the e-Polis, In: *Information Security and Ethics: Concepts, Methodologies, Tools, and Applications*. H. Nemati, Ed.

[22] Jägemar, M. and Dodig-Crnkovic, G. 2015. Cognitively Sustainable ICT with Ubiquitous Mobile Services - Challenges and Opportunities. In *Proc. Int. Conf. on Software Engineering (ICSE 2015)* Florence. Italy.

[23] Larsson, M. 2004. Predicting Quality Attributes in Component based Software Systems. PhD Thesis. Mälardalen University Press, Sweden.

[24] Larsson, S. 2005. Improving Software Product Integration. Licenciate Thesis. Mälardalen University Press, Sweden.

[25] Kade, D. 2014. Towards Immersive Motion Capture Acting Design, Exploration And Development of an Augmented System Solution. Licentiate Thesis. Mälardalen University Press, Sweden.

[26] Computer Science 2013: Curriculum Guidelines for Undergraduate Programs in Computer Science. The Joint Task Force on Computing Curricula. Association for Computing Machinery (ACM) IEEE Computer Society

Author Index