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).

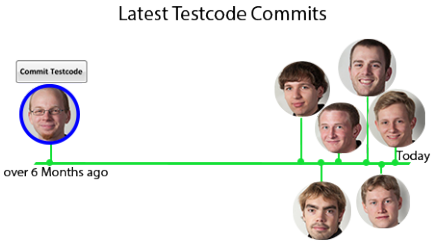

Figure 1: Visualization of the testing signal 'Latest Test Code Commits'. It shows the team's time line of test code committing.

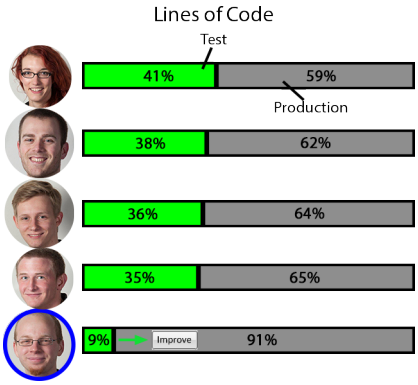

Figure 2: Testing signal 'Test Code Ratio' displays the team's ratio of test code vs. production code.

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.

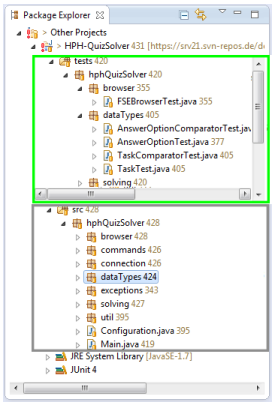

Figure 3: Testing signal 'Test Code Explorer' displays test classes more prominently in the IDE.

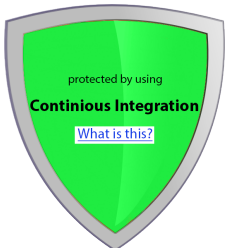

Figure 4: Testing signal 'Using Test Services' displays the use of testing tools in a project and emphasizes the team's commitment to systematic testing.

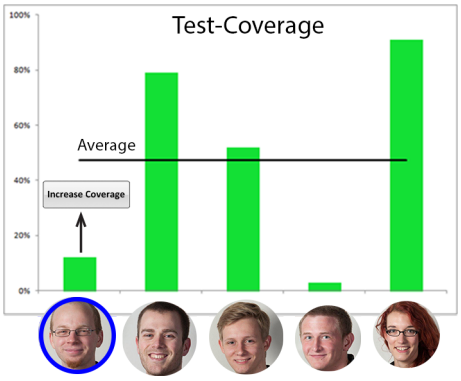

Figure 5: Testing signal 'Test Code Coverage' shows the team's effort to cover their code with tests.

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'.

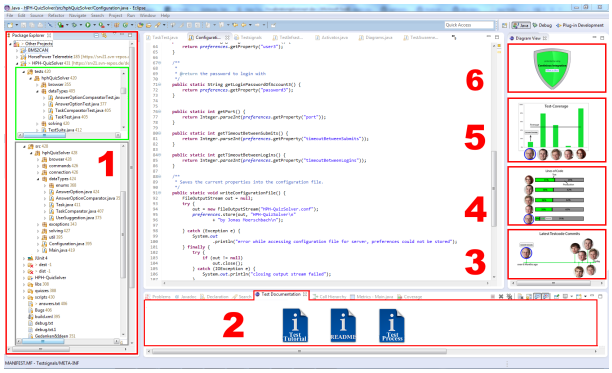

Figure 7: All six testing signals integrated in Eclipse: 'Test Code Explorer' (1)', 'Test Code Documentation' (2), 'Latest Test Code Commits' (3), 'Test Code Ratio' (4), 'Test Code Coverage' (5) and 'Using Test Services' (6)

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.