Creating a Shared Understanding of Testing Culture on a Social Coding Site

Raphael Pham*, Leif Singer*, Olga Liskin*, Fernando Figueira Filho†, and Kurt Schneider*
* Leibniz Universität Hannover
Hannover, Germany
{firstname.lastname}@inf.uni-hannover.de
† Universidade Federal do Rio Grande do Norte
Natal, Brazil
fernando@dimap.ufrn.br

***Abstract*—Many software development projects struggle with creating and communicating a testing culture that is appropriate for the project's needs. This may degrade software quality by leaving defects undiscovered. Previous research suggests that social coding sites such as GitHub provide a collaborative environment with a high degree of social transparency. This makes developers' actions and interactions more visible and traceable.**

We conducted interviews with 33 active users of GitHub to investigate how the increased transparency found on GitHub influences developers' testing behaviors. Subsequently, we validated our findings with an online questionnaire that was answered by 569 members of GitHub. We found several strategies that software developers and managers can use to positively influence the testing behavior in their projects. However, project owners on GitHub may not be aware of them. We report on the challenges and risks caused by this and suggest guidelines for promoting a sustainable testing culture in software development projects.

I. Introduction

In recent years, social media have changed how software is developed. Software engineers connect with, provide help to, collaborate with, and learn from one another with unprecedented ease [1]. Blogs and Q&A sites like Stack Overflow foster collaborative learning, covering substantial amounts of official API documentation [2]. Social coding sites like GitHub act as version control repositories with a Web interface, as well as a social network site [3]. Finally, developer profile aggregators like Masterbranch and Coderwall combine the activities of developers to generate unified profiles of software developers that help them discover new contacts and technologies [4].

Many social media sites — both general and those specifically created for software developers — provide a high degree of social transparency [5]. Members are able to easily find out who they are interacting with, whom everyone else is interacting with, and who has interacted with which artifacts. This transparency influences the behavior of software developers, as documented by Dabbish et al. for GitHub [3]. For example, developers assess each other using social media and are aware of the publicity of their own activities, therefore taking care of how they behave. One of the topics mentioned by Dabbish et al. is *software testing*.

This paper investigates how testing behavior is influenced by the peculiarities of social coding sites like GitHub. Different testing practices may influence software quality, maintainability, and development times. For example, in some cases test-driven development (TDD) has been shown to significantly lower the defect rate of software products [6].

If social coding sites can influence the testing behavior of developers, they might also have an impact on the progress of software development projects and their resulting products. Understanding these influences better might enable individual developers and software development organizations to positively influence their own teams' testing practices.

As we were not able to find previous work on this topic, we conducted an exploratory study of testing on GitHub to find out how social coding sites influence testing behavior, which challenges this creates, and how developers cope with these challenges. With our approach based on Grounded Theory [7], we conducted interviews with 33 active members of the GitHub social coding site. We answered our research questions by analyzing these interviews and validating our results with an online questionnaire that was answered by 569 active GitHub users.

Our results suggest that there are several strategies that software developers and managers can use to positively influence the testing behavior in their projects, but that project owners on GitHub may not be aware of them. For example, some participants of our study reported that existing tests in a project would influence them to include tests in their commits — however, the developers receiving such contributions saw no such connection.

This paper is structured as follows. Following this introduction, we provide a short overview of the terminologies of Git and GitHub. In section IV, we document our study design; we report our findings in the following two sections. Section VII provides a discussion of our findings, while section VIII lists the limitations of our work. Section III discusses related work. Section IX concludes this paper and gives an outlook to future research.

II. Git and GitHub

This section provides a short introduction to Git and GitHub, and introduces some of the terminology used in the remainder of this paper.

978-1-4673-3076-3/13 © 2013 IEEE

ICSE 2013, San Francisco, CA, USA

Git[1] is a distributed version control system (DVCS). Each contributor may *clone* a remote repository to create a local copy of the whole repository and may then *commit* changes to this local repository. Changes can be *pushed* or *pulled* to and from remote repositories. Often, there is one remote repository combining the participants' commits.

To avoid having to grant every *contributor* full commit rights to that central repository, developers often use a process coined *forking*. Contributors create clones of the central repository and, eventually, if they want a commit to be included in the central repository, they issue a *pull request*: a notification to the *project owner* — the manager(s) of the central repository — that new commits are available on the developer's fork. The project owner may then decide themselves whether to pull those changes to the main repository. This decouples outside contributors from the central repository. For simplicity, we will refer to the project owner in the singular form, even though several developers might have that role for a project.

The social coding site GitHub[2] provides hosting of remote Git repositories and supports the repository interactions in a Web-based user interface. Pull requests can directly be commented on, thus facilitating discussions. GitHub integrates tools that are often used in software projects, such as an issue tracker or a wiki. Because of this easy accessibility and a streamlined contributing process, projects hosted on GitHub are accessible to a large number of potential collaborators.

Each member maintains a profile site, may *follow* the activities of other members, and may view the contact lists of other members — making GitHub a social network site as defined by Boyd and Ellison [8]. Thus, project owners and contributing team members are easily reachable for potential external contributors.

III. Related Work

This section gives an overview of research regarding open source (OSS) software development, related testing activities and social coding sites.

Testing Practices in Development Communities: Several surveys report on testing practices found in different communities. For example, Greiler et al. [9] investigate the testing practices in the Eclipse Plug-In community. Michlmayr et al. [10] report on a study that investigated the quality-related practices in a diverse set of open source projects and associated problems. To the best of our knowledge, testing practices on social coding sites have not been examined before. Regarding testing on GitHub, we present a first understanding of the interactions and motivation when contributing to a project or maintaining it.

Social coding sites with their high degree of social transparency have recently gotten into the focus of research. Dabbish et al. [3] investigate the influence transparency has on the behavior of participants of GitHub. Stuart et al. [5] provide a framework for thinking about social transparency and its effects. Singer et al. [4] examine developer profile aggregators that combine developers' activities from social coding sites and other social media. They find that this *social programmer ecosystem*, as they call it, helps developers connect, collaborate, and discover new technologies. However, none of these studies has examined the effects increased social transparency might have on testing practices in projects. Besides complementing to further understanding the dynamics in social coding sites, we also provide suggestions for guidelines on how to communicate testing requirements when collaborating on GitHub.

Open Source Software Development: Some of the phenomena observed on GitHub are typical for open source software development, where a lot of research has been conducted. For example, Mockus et al. [11] characterize OSS development and describe its development process based on the Apache Web Server project. Bonaccorsi and Rossi [12] as well as Krishnamurthy [13] investigate the motivations of individuals and software companies that participate in open source software development.

Further, there is substantial research on *coordination* in open source projects. Bird [14] investigates how developers coordinate in large OSS projects. Crowston et al. [15]–[17] investigate coordination, task assignment and communication in different case studies. One of their findings is that the most prominent mechanism is self-assignment of tasks. In one study, Crowston and Scozzi [18] explicitly study bug fixing practices in OSS development. Stewart and Gosain [19] show how ideology in an OSS project influences its effectiveness. Dagenais et al. [20] investigate the creation and maintenance of developer documentation and how developers use it to learn in open source environments. Sowe et al. [21] describe how knowledge is shared and used via mailing lists.

While these studies report on phenomena that we also found in our work, our contribution adds to them by discovering influences up to now only found in our study. We believe that this is due to peculiarities of social coding sites, such as the high degree of social transparency, remarkably low barriers, and a high centralization and integration of tools and interfaces.

Coordination and awareness within teams are the subject of some CSCW research focused on software development. For example, Begel and Zimmermann [22] report on a newsfeed system that increases project awareness. DeSouza and Redmiles [23] investigate which actions should be presented to whom in such feeds. GitHub also uses mechanisms like newsfeeds, increasing social transparency. However, whether and how such mechanisms influence testing behavior had not yet been examined in this field.

IV. Study Design

We used an approach based on Grounded Theory (GT) [7] to explore how testing is carried out on social coding sites and how an increased social transparency impacts the testing behavior of software developers engaged in those projects. For the purposes of this paper, we focused on two groups

[1] http://git-scm.org
[2] http://github.com

of people: (1) *project owners*, who receive pull requests and (2) *contributors* who send them.

A. Research Questions

We designed a set of research questions to better understand how testing practices evolve based on the interaction between contributors and project owners on social coding sites. First, we focused on identifying the main steps and variations of the contribution process and how decisions are made with regard to testing. This leads to our first research question:

RQ1: What is the contribution process with regard to testing on social coding sites?

While investigating the contribution process on GitHub, it became clear that contributions were assessed by project owners. Furthermore, we found different motivations for implementing quality assurance measures specific to interaction on social coding sites. We refined our first research question into two sub questions to accommodate this.

RQ1.1: How do project owners assess incoming contributions from contributors?
RQ1.2: What are the internal and external motivations for engaging in testing efforts on social coding sites?

As reported by Stuart et al. [5], social transparency may have hard to anticipate second order effects in addition to the possibly intended first order effects. In our study, we wanted to find out more about the issues that might arise as a result of those effects. As such, our second research question is:

RQ2: What challenges and risks related to testing arise from engaging in projects on social coding sites?

In face of such challenges and risks, project owners and contributors might have to evolve and adapt their contribution process. For example, project owners might have to advise contributors on how to conform to project guidelines. Our third research question focuses on the actions taken by each group of actors for overcoming such issues:

RQ3: How do developers cope with those challenges and risks related to testing?

Finally, the increased social transparency on social coding sites creates distinct modes of interaction between project collaborators. Understanding these modes of interaction can help guiding practitioners as to which kind of interactions to support and which to avoid in the purpose of improving the existing testing practices. This leads to our last research question:

RQ4: What impact does the participation on social coding projects have on testing practices?

B. Procedure

Grounded Theory [7] emphasizes a continuous data collection process interlaced with periodic pauses for analysis. As such, we conducted our study in three phases.

First, we focused on understanding which testing-related norms and conventions exist on GitHub. For our investigation, we obtained 16,000 email addresses of recently active users by querying the GitHub Archive[3]. From this pool, we invited 50 users to semi-structured interviews and another 50 users by randomly choosing a member from each of the 50 most successful GitHub teams as listed on the Coderwall[4] leaderboard. This sampling strategy resulted in a diverse population: highly experienced as well as regular users of GitHub.

Of these 100 users, 10 from the former sample and 3 from the latter sample enrolled. Each participant was interviewed by a member of our research team via voice call. Interviews lasted approximately 20 minutes and the audio was recorded. We asked the participants to outline the testing process in one of their public projects on GitHub. Preliminary findings indicated that projects featuring extensive collaboration between developers would demand more elaborate testing approaches. We also learned that most decisions regarding testing were made when pull requests were sent and received, i.e., when people had to coordinate and negotiate their cooperation.

In the second phase of our research, we defined our target population to be active users who used the collaboration features of GitHub. From our address pool, we invited 1,000 GitHub users at random to take our first questionnaire (Q1): 500 at first and again 500 after 6 days. In total, 158 users responded, of which 74 left usable contact information. The questionnaire responses allowed us to distinguish between users that had been collaboratively active — they had either sent a pull request or forked a project — and their approaches for testing contributions. We invited all 62 of those 74 who matched our criteria for another round of semi-structured interviews. 20 users enrolled.

In these interviews, we inquired about participants' testing practices and values. We had prior answers to these questions from questionnaire Q1 and used the interviews to explore such situations in detail, allowing us to better understand the contribution process with a focus on testing behavior and practices. For example, we asked interviewees how they *handled incoming pull requests* and whether they had any kind of quality assurance measures related to this process. Then, we inquired about their motivations for assuring quality and the challenges they face when *contributing* to other projects. As a result of our second phase, we were able to identify five themes that stood out.

1) The fork and pull request mechanisms, social network features, and integration of numerous tools result in a *GitHub-specific process* for sending and receiving contributions.
2) GitHub makes it easier to access a public repository, start working on it, handle contributions, and discuss them with contributors. GitHub tools and social features *lower the barriers for engagement* in software projects.
3) Public projects and profiles on GitHub have *high exposure* to many potential contributors and users. This helps with, for example, discovering edge cases.
4) GitHub *integrates* many tools into the project con-

[3]http://www.githubarchive.org
[4]http://coderwall.com

text and *centralizes* many interactions and notifications among project participants.

5) GitHub provides increased *social transparency* that allows its users to see the identity, actions, and communications between users, a phenomenon that was previously reported on by Dabbish et al. [3].

In the last phase, a final questionnaire (Q2) was sent to 4,000 random GitHub users for quantitative validation of our findings. Of these users, 569 responded. The results of this phase can be found in section VI.

C. Data Analysis

We alternated periods of data collection with analysis in order to build up our theory. At the end of each data collection phase, we transcribed the recorded interviews and open coded them. This resulted in 172 codes. At the end of our second phase, we engaged in axial coding our codes in order to find higher level conceptual themes that would help us in answering our research questions. In the last phase of our research, core themes of our theory were formed into statements and validated through a final questionnaire (cf. section VI). Participants were asked to agree or disagree with statements using a Likert-type scale. For each question, the mean value for the given set of answers was calculated, as well as its variance and the number of given answers to that question in total. For questions that required the participant to choose an answer in a set of pre-given answers, the count for each answer was calculated and related to the total number of answers to that question.

D. Participants

Overall, we interviewed 33 people, among them software developers, testers, and software architects. Nearly half of them were using GitHub for professional work (19); the other half (14) used GitHub for private projects. Our population comprised of developers employed in a software company (24), self-employed (3), and unemployed developers (2). 4 interviewees were affiliated with universities and mostly engaged in noncommercial projects. 20 participants of our second interview phase estimated the number of total contributors to their projects. Numbers of contributors were diverse: 12 projects with up to ten contributors, two projects with up to one hundred contributors and six projects with over one hundred contributors. We denote these interviewees with *FI*. Randomly chosen interviewees from the first interview round are denoted with *R*, Coderwall leaderboard members are denoted with *L*.

V. Findings

A. Interaction on GitHub with Regard to Testing

This section reports on our findings regarding the contribution process on GitHub and how project owners assess pull requests from contributors (RQ 1.1).

In the course of our interviews, several steps of the contribution process on GitHub emerged. After receiving a pull request, the first step that project owners conducted was to manually review the contribution and assess it by different aspects. After this review, they merged the pull request into a testing branch and resolved conflicts manually. Superficial adjustments like code style corrections or comments were added based on preference. If a test suite existed, project owners ran it to check whether or not this contribution passed tests. This increased confidence in the contribution. Finally, the contribution was merged into the main branch of the project.

We found many factors that were taken into consideration by project owners when assessing contributions. For instance, project owners reported to treat incoming pull requests differently depending on whether they ***trusted*** the contributing developer. Pull requests from unknown developers would undergo a more thorough assessment, while contributions from trusted developers would be merged right away. *"if it's someone I trust, who's worked on the project a lot, then I don't do that. [...] if it's someone who hasn't spent a lot of time on the project, I'll try and do that." [L48]*

The perceived ***size of the changes*** highly influenced the project owner's need for tests. If the project owner believed to have quickly understood the changes' impact, they demanded no tests from the contributor. This was often the case when only some lines of code had been changed.

Additionally, project owners distinguished between two ***types of contributions***: contributions that introduced a new feature or contributions that changed existing code, such as bug fixes. The former were requested to include tests, while the latter caused project owners to check whether or not the changed code was already covered by existing tests. If so, no further tests were demanded. *"if you really write a new feature, then it makes more sense to add tests for that, but if you just do a little change in a code chunk that is already there then I don't expect that the person writes the test for that." [FI19]*

The ***target of the changes*** was considered as well. Changes to core functionality caused a demand for tests. However, if the ***estimated effort*** for creating automated tests was regarded as infeasible, this demand was waived. Often, such tests would require a cumbersome setup of test environments (such as different operating systems). Project owners were aware that contributors acted voluntarily and were unfunded in most cases.

B. Motivations for Demanding and Delivering Tests

This section presents our findings regarding project owners' motivations for demanding tests and contributors' motivations for providing them (RQ 1.2).

Project owners' reasons for demanding tested contributions were manifold. Maintaining clean and well documented code reduced the subsequent ***support effort***, according to one project owner. Some project owners perceived tests as a form of ***documentation*** of how to use the contributed feature. Another project owner requested contributors to provide test cases of how they needed the software to behave, so he could merge these into the existing test suite for future regression testing. In this case, tests were used for ***communicating requirements***.

An external motivator was the impression of acting as a ***role model*** when working on a testing-related project. Several users reported to feel obliged to perform proper quality assurance for projects with a ***domain related to testing*** — e.g. a testing framework or a continuous integration server.

In interviews with contributors, different motivations for including tests in a pull request on one's own initiative emerged. Some interviewees said they explicitly added tests that ***highlight the value*** of their contribution to the project owner: these tests failed with the old version of the software in question, but passed when the contribution was applied.

The ***existence and prominent placement of tests*** gave contributors the impression of informal project guidelines; thus they felt obligated to add their own tests. *"[There were no guidelines set up], not so much formally, but it was pretty clear how it was supposed to be tested and there was already an existing spec file [...] with a pretty substantial list of tests" [FI17]*

Similar to a project owner perceiving oneself as a role model when working on a testing-related project, contributors felt an ***implicit demand*** for tested pull requests in such domains. In other instances, this obligation also resulted from customs rooted in the community of the technology used. Often, Ruby developers tested their contribution by default.

C. Challenges and Risks

This section discusses the challenges and risks we found to arise when engaging with projects on GitHub (RQ 2).

Interviewees saw an ***urgent need for automatic tests*** in their projects. This need was felt very strongly, as there are a lot of contributors on GitHub which are often only marginally engaged, i.e., there is a large group of developers in the periphery. Therefore, a lot of contributions need to be managed which interviewees reported could not be done using manual tests — simply for ***reasons of scale***. *"a lot of people are contributing to [the project] and quality control is becoming more and more important to us. Automated testing is the only way to get that" [FI7]* To achieve automated testing, project owners were in many cases looking for tests when they received pull requests from contributors (cf. the previous section).

Because there is a ***constant flux of contributors***, and developers new to a project are not yet accustomed to its testing culture, they have to learn it anew. This takes time and effort. If a project fails to ***communicate testing culture*** efficiently and effectively, or sets ***barriers that are too high*** for first-time contributors, it can struggle to create such a culture. For example, if new contributors cannot easily find ***existing tests*** in a project, they will not be able to write any of their own for their contributions. *"When I first started contributing to [the project], they did not actually have a test suite which made shipping them tests fairly difficult" [FI20]*

Several project owners reported that their projects were ***struggling with creating the required testing culture***. The pull requests they received would often not include any tests by default. One issue they saw was the ***voluntary nature of open source*** contributions: they could not simply require well-tested contributions. Developers who had sent a valuable pull request might be alienated if project owners rejected their contributions due to a lack of tests. *"We have a project, where we don't have the culture, it is difficult and people are volunteers, we can't just enforce it on the project. You have to try to incubate it into the project." [FI7]*

Another reason why creating a robust testing culture is so difficult could be the lack of experience on the side of the contributors. *"We have a lot of people ramping up to the team every time. So, we have a big rotation. So new people don't understand what is a quick build, what is the regression, what is isolated, ... and they don't understand how to write tests for each of those suites." [L4]*

Two of GitHub's greatest strengths — low barriers to contribution, combined with tight integration of related tools and services — are related to another challenge that interviewees mentioned. In terms of testing, there are ***no integrated tools*** provided by GitHub that might lower some testing-related barriers, e.g. for setting up a server for continuous integration. Even though this is starting to be supported by Travis CI[5], this was not yet commonly used amongst the interviewed population. One interviewee even mentioned that *"github its such an easy-to-use-tool that it makes writing unit tests seem like extra time for most people." [R16]*

D. Coping with Challenges

In this section, we report how interviewees coped with the testing-related challenges on GitHub (RQ 3).

As described in the previous section, scalability reasons drive project owners towards automated tests. However, manually merging each pull request into a testing branch and running regression tests with a test suite of automated tests remained a tedious task. Some interviewees resorted to an ***automated continuous integration*** (CI) service, such as Travis CI or Jenkins. Such a service frees the project owner of several manual steps: when a pull request is received, a program merges the contribution into a testing branch, runs the existing test suite, and notifies the project owner as well as the contributor of the results.

Project owners developed different strategies to establish a common understanding of testing requirements and handling untested pull requests. Due to the voluntary nature of GitHub (c.f. previous section), some project owners simply resorted to ***writing tests themselves*** or ***thankfully requesting*** — instead of demanding — further tests, as this leaves the contributor the option to decline.

Lowering the barriers and making it ***easier to provide tests*** was another strategy, for example by introducing a testing framework with a suitable explanation of its usage. Other project owners provided ***easy access to learning resources*** and ***actively supported*** contributors who showed difficulties in writing tests: pointing contributors to tutorials, suitable examples in an existing test suite, or actively teaching them.

[5]http://travis-ci.org

This was beneficial as contributors were convinced of the need for tests and included tests on their own in subsequent pull requests. *"[...] I can point them to one of the existing tests. «Check this one, it is really similar to what you need», and in most cases it's enough. Sometimes, [...] I do Skype conferences with screen sharing where I can explain, show." [FI4]*

A passive strategy for communication testing requirements was to make it more ***obvious*** that testing was indeed desired by giving the impression that testing was a customary in one's project. Some interviewees said that they tried to ***lead by example*** and tried to make testing visible in the project, hoping to engage contributors in testing. Another strategy was to ***display testing signals***. For example, every project using Travis CI may add a badge to its profile page. This informs a potential contributor that continuous integration is regularly performed. *"if you see that image, it immediately rings the bell that there is continuous integration in this project, and as such, there is some kind of automated testing." [FI6]*

E. Impact of Social Coding Sites on Testing Practices

This section investigates the impact of engaging with a social coding site on testing practices (RQ 4).

During our research, we encountered different levels of lowering the effort for testing. One interviewee suggested that in his perception, projects using behavior driven development (BDD) and concentrating on testing primarily the ***main use cases*** provided these low barriers to entry. This allowed those projects to deliver results very fast, which impressed other developers. Those projects were also very good at communicating their culture in social media, e.g. via blog posts advocating their testing culture. This combination, the interviewee said, would lead to the described behavior being adopted at a fast rate and thus spreading through GitHub. He saw this development critically however, as possibly important edge cases were ignored until they became apparent.

A more extreme case of lowering the barrier for contributions was simply to ***defer testing*** to a later stage of the project. An interviewee said that in his experience, younger projects needed to gain traction in the present. Obtaining external contributions was regarded to be more important than quality assurance measures. However, such projects would need to pay back this accumulated technical debt in the future.

Interviewees told us that effectively ***communicating a project's testing culture*** would lead to contributors adopting that culture more rapidly and in greater numbers. We spoke to several developers who had experiences in multiple projects with different communication strategies. According to them, better communication of testing culture leads to more pull requests containing tests, and therefore to projects that were tested better. ***Testing guidelines*** and ***actively communicating*** to contributors what kind of tests were required helped getting more contributors to provide tests with their pull requests. Providing guidelines on contributing or providing dedicated testing tutorials removed uncertainties in contributors about how to participate correctly. This way, the barrier of having to ask was removed. One interviewee expressed that knowing a project's testing culture and adhering to it would make him and his peers feel ***proud***, further helping the project's testing culture to be adopted. However, the reverse was also mentioned: if a project did not communicate anything about its norms and requirements regarding testing, new contributors would simply ***assume that no tests had to be written*** and consequently submit pull requests not containing any.

Some interviewees mentioned that testing culture — and communicating it — could even be ingrained not only in a single project, but in a whole ***infrastructure community*** (c.f. section V-B). According to these interviewees, in the Ruby community it was taken for granted that blog posts, screen casts, and tweets from popular developers often talked about testing practices that were regarded as proper in that community. This is a form of communicating a testing culture — in this case however, in a much larger realm than that of one or more GitHub projects.

Interviewees reported several examples where ***direct exchanges*** on GitHub helped diffusing testing culture. For example, one project owner reported that his project started using the Travis CI service when it received a pull request that added a Travis CI configuration file. *"I received a Travis CI config [in a pull request]. I did not know this service before and someone send me a config for Travis and this is how I came to use it" [FI19]*

Travis CI, in turn, also arranges for ***low barriers*** and ***easy communication*** of testing culture. As described in section V-D, the Travis CI badge is perceived as a signal that communicates the use of certain practices.

However, employing a continuous integration service with an extensive test suite may create a ***false sense of security***: one interviewee reported to use the positive result of running his existing test suite as a sufficient confirmation of a contribution's correctness. He usually simply merged such pull requests without requesting any additional tests specific to these contributions.

A key instrument for project owners wishing to create and nurture their project's testing culture was to provide existing tests and a testing infrastructure that was easy to set up (c.f. section V-D). Interviewees reported this to ***lower the barrier*** to accommodate to the project's culture. They would just need to fork the project, execute a shell command, and have the existing tests running. Interviewees took this as an opportunity to run regression tests, thus trying to make sure that their contribution did not break anything. This increased their confidence in the correctness of their own code and lowered the barrier of contributing. Some interviewees also said that just having a test suite would ***communicate certain values*** regarding testing, helping them understand the project's testing culture, norms, and conventions. Providing ***publicly available tests*** brought by another advantage: contributors heavily used existing tests as a source for education and examples for their own test cases. *"there were some tests surrounding some quite similar functionality in the source, so I basically copied and modified these tests to test the functionality that I added." [FI20]*

One recurring theme was that communicating of testing culture better and lowering barriers in a project ***promote exploration and experimentation*** for new contributors. Interviewees claimed that this, in turn, often leads to developers becoming more familiar with a project's testing culture, making it easier for them to provide their own tests. For example, a project with a CI server that provided fast feedback on tests was said to support experimentation — after all, problems would be easily visible, giving experimenting developers more confidence. Some interviewees mentioned that experimentation was supported not only by a project's deliberate efforts, but also by the chosen programming language and the available libraries.

Interviewees claimed that the number of volunteer contributions increased since moving their projects to GitHub. They attributed this to the low barriers to entry and the resulting exposure to a larger number of developers. An employee of a company that develops open and closed source projects told us that many GitHub contributors find bugs, provide tests, and send them bug fixes. This, he claimed, enabled the company's paid developers to ***concentrate on larger issues***, such as creating new features.

To a greater extent, interviewees reported that the ***public nature*** of software development on GitHub leads to an improvement of testing practices. Some of them were companies that used the exemplary testing practices in their public projects as an ***advertisement*** for the high quality of their development services. Indeed, one employee of such a company confided that testing was less important in the company's internal projects, as they did not serve as such advertisements.

We heard similar reports from open source projects not backed by companies. Core members of such projects were concerned about the project's ***reputation*** — how the project was perceived by the community. They believed that proper testing would lead to higher quality code, which in turn would be received better by others. *"we need to have a more substantial testing framework because it's [...] a significant indicator of code quality in the community. If you don't have good tests then people start to suspect that your code may not be any good either." [FI17]*

On the side of the contributors, one interviewee reported that having contributed to a high-profile project in which tests are mandatory would help him ***find work***. Indeed, that was his only reason for contributing.

Our findings show that several mechanisms and processes used on GitHub may help projects become better tested. Project owners help new contributors get acquainted with a project's culture and make it easy for them to get up and running technically. However, this does not only affect regular contributors that can be productive faster. ***Drive-by commits*** — as an interviewee called them — are small changes that do not require a prolonged engagement with a project, yet provide some value for it. Developers providing such changes would not always be actively interested in a project, but might have stumbled upon it when browsing GitHub. Then, when they had found, for example, a spelling error or a missing translation, they would make a quick correction and forget the project again. GitHub's integration and low barriers seemingly streamline this process and might lead to GitHub having a very long tail of many small contributions.

VI. Validation of our Findings

In section V, we presented findings gathered by conducting 33 interviews with GitHub users. This section presents the most striking results of our final questionnaire (Q2) that we used to validate core statements of our findings. Of 4,000 random GitHub users, 569 filled out this questionnaire. Our results are summarized in Table I. *PO* denotes statements about project owners, *C* about contributors. For each question, we presented the participant with a Likert scale of five items ranging from "I do not agree at all" to "I strongly agree.". These Likert items are represented in the diagrams on the X-axis with disagreement being on the left and agreement being on the right hand-side. The middle bar represents the neutral item. The number of answers accumulated for each Likert item is marked on the Y-axis. The grey bar is the median.

The questionnaire required the participant to take both the perspective of a project owner and a contributor. 39% of our participants would receive pull requests at least a few times per week (daily: 16%) and 27% send pull requests at least a few times per week (daily: 6%). Even though several interviewees mentioned voluntarism as a hindering factor, the questionnaire did not validate this (PO1). Personal traits such as modesty and humility of the requester as well as value given to the contribution may be influencing factors. Similarly, most of the participants did not agree to feel a need for automation (PO2). However, as some interviewees mentioned, this need may depend on the size and popularity of the project in question. As popularity grows, the amount of incoming pull requests increases. As both samples were randomly invited, but ultimately self-selected, variations in populations might attribute for this dissonance.

Project owners agree that providing tests in one's project lowers the support effort regarding testing by contributors (PO3). Appropriately, contributors heavily rely on existing test cases when creating their own. They use these to learn how testing is done in a specific project (C3) and, lastly, even copy existing tests and use them as a basis for new tests (C4). With tests in place, contributors feel obligated to add their own tests and seem to comply with this implicit demand (C2).

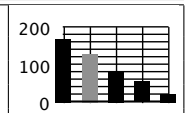
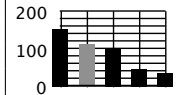
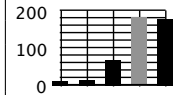
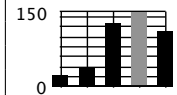
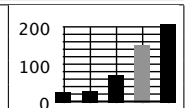
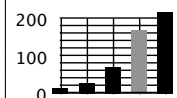
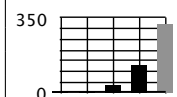
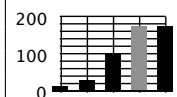
VII. Discussion

This section relates our work to previous research and discusses its potential impact on the software development industry, open source development, and research. We connect our findings with research in Communities of Practice and the diffusion of innovations. In doing so, we distinguish between processes occurring inside of projects — intra-project — and those spanning multiple projects, i.e., inter-project processes.

A. Creating a Shared Understanding

Lave and Wenger [24] coined the term *Community of Practice* (CoP) for groups of individuals that work on similar problems and exchange knowledge about good practices,

TABLE I
RESULTS OF THE FINAL QUESTIONNAIRE (Q2).

Statement and Question	Results
PO1:Voluntarism hinders demand of tests. I have the feeling that I am not in position to demand tests from a contributor as he or she is contributing on a voluntary basis. (464 answers)	
PO2: Amount of incoming pull requests demands for automation. The amount of incoming pull requests is so big that I can only assure their quality by using automated tests. (453 answers)	
PO3: Existing tests support contributors in writing their own tests. When I have tests in my project, contributors need less help in writing tests. (457 answers)	
PO4: Existing tests facilitate more incoming pull requests that are tested. As a consequence of providing tests in my project, more pull requests include tests. (452 answers)	
C1: Low barrier commit mechanism facilitates Drive-By-Commits. Since it is so easy to send a pull request, I contribute more changes that I would not have engaged in otherwise. (496 answers)	
C2: Existing tests make contributor feel obligated to add tests. When I see that there are tests in a project, I will also include tests in my pull request. (499 answers)	
C3: Existing tests are a source of education for contributors. Existing tests help me in understanding how to test in a specific project. (495 answers)	
C4: Contributors use existing tests as a basis for new tests. I use existing tests as a basis for my own tests: I copy and paste them and adjust them accordingly. (496 answers)	

proven solutions, etc. with each other. They mention *legitimate peripheral participation* as a central phenomenon when describing how new members of the CoP join and, through learning the community's norms, become more and more involved with it. Initially, novices merely observe practices passively before starting to take on simple and increasingly complex tasks. New members are said to be situated in the community's periphery, while established members are part of the core. As shown by Crowston et al. [16], these processes can apply to open source software development as well.

1) Intra-project Processes: Inside of individual projects, we found that new contributors start off with first observing how pull requests are handled and discussed, and what good commits and tests look like. This is supported by the high social transparency found on GitHub.

When they are ready to submit their own pull requests, they have already learned quite a lot about the project's testing culture. However, they are often assisted further by the low barriers many projects on GitHub strive to provide to potential contributors. Examples for this are existing tests that can be simply copied and modified and the fact that several project owners told us that they strive to provide testing infrastructure that is easy to set up. This is again supported by test automation integrated with GitHub itself, such as Travis CI.

This increased level of support for peripheral contributors seemingly creates very large peripheries of contributors for projects, as touched upon in some of our interviews. Consequently, some of the mechanisms we found may be used solely for managing peripheries of this greater size. Because of the exploratory nature of this work, we were not yet able to gain deeper insights about the properties of such projects. Future research will need to investigate the problems created by projects with such compositions and how project members manage these challenges.

Drive-by commits — simple commits that leave their creators rather uninvolved with the project and that can be created with very little project-specific knowledge — are a departure from the model of the peripheral member that gradually gets more involved with a community. We believe that more research is needed to help us understand the motivations and processes surrounding this phenomenon better.

2) Inter-project Processes: In addition to these phenomena related to individual projects, some interviewees told us about how the testing culture of communities works that span multiple disconnected projects. Most of the time, these would use the same programming language and the same frameworks for development.

For example, the Ruby community seems to have a distinguished testing culture that many interviewees were aware of. Core members of the community create the frameworks that more peripheral members will use, and also publish learning resources such as blog posts and screen casts. In these frameworks and documents, they advocate a certain testing culture: e.g., behavior-driven development (BDD), supported by BDD testing frameworks, and characterized by a focus on testing primarily the *happy path* — the intended behavior, ignoring edge cases for a large part.

As one interviewer opined, frameworks and the respective testing culture allow such projects to move very fast and to produce impressive results more easily. He argued that this was part of why the testing culture gets easily adopted by novice community members.

B. Diffusion of Testing Practices

GitHub does not only help Communities of Practice create a shared understanding of their respective testing culture in peripheral and novice contributors. It also facilitates the diffusion of these practices among developers inside and outside of individual communities.

Research on the diffusion of innovations investigates how and why tools, practices, ideas, or technologies perceived as new — *innovations* — are adopted by individuals and groups. Rogers [25] documents properties of innovations that were

discovered to support their adoption across many different scenarios.

- *Relative Advantage:* adoption is more likely if the innovation has a clear advantage with regard to known alternatives.
- *Compatibility:* the more compatible an innovation is to a person's existing practices, the more likely it is that she will adopt it.
- *Complexity:* the more complex an innovation is perceived to be, the less likely it will be adopted.
- *Observability:* an innovation will be more likely to be adopted the easier it is to observe existing adopters.
- *Trialability:* the easier it is to try out an innovation before deciding to adopt it, the more likely it is to be adopted.

We now relate these properties to the phenomena we found in our work.

1) Intra-project Processes: Positive results of testing practices, such as adding features fast or badges with passing test results, demonstrate the *relative advantage* of those practices. As project owners strive to make it easy for new contributors to get started with their project and its test suite, they actively improve their project's *technical compatibility* with developers' existing practices. In the same vein, by communicating desired testing behavior and aligning it with the values promoted by thought leaders, they improve their project's *cultural compatibility*. By lowering the barriers to entry — e.g., by providing existing tests, examples, and a working infrastructure for automated tests — project owners reduce the perceived *complexity* of their project's testing practices. Several integration and user interface features of GitHub support this, such as the built-in ticket system, external services like Travis CI, or the comfort with which code can be inspected using a Web browser.

Lowering barriers also increases the *trialability* of testing practices. If developers want to try a project out themselves, all they have to do is clone the repository to their local machine using a single command. For certain communities, one more command will install all dependencies and run the project's tests: one interviewee noted how the Ruby community makes this process especially easy.

Finally, the social transparency on GitHub makes testing practices more *observable*. By looking at commits, issues, pull requests, and the respective discussions surrounding those items, developers on GitHub are able to observe the testing culture in a project without needing to become involved much.

2) Inter-project Processes: As reported by several interviewees, they also use GitHub to discover new projects, and to learn more about those they already know about. Being able to follow the activity of developers and to browse projects by technological niche support this discovery. In this regard, the aforementioned properties of GitHub apply not only to individual projects, but also to the diffusion of testing practices *across* projects. For example, one interviewee told us that he uses GitHub to learn how other projects use the testing framework that his work project uses (L4). This shows how these mechanisms are even able to diffuse practices into organizations not necessarily hosting their projects on GitHub (which was the case for L4).

C. Impact

By connecting our results with established theoretical models, we document the context of our research and thus make it more accessible to individual software developers, software development organizations, and researchers. This section outlines how our results could be built upon by them.

1) Software Developers and Organizations: We discovered several mechanisms that help creating a shared understanding of a project's testing culture and diffuse testing practices to other individuals and groups. Because of the exploratory nature of our research, we have not yet discovered best practices, but candidates for such. From these mechanisms, we can derive several preliminary guidelines.

Companies and core members of software projects should strive to *lower the barriers* to testing by providing testing guidelines, test examples, an easy to set up testing infrastructure, and integrated automatic testing. This should support cultivating a project culture that embraces appropriate testing practices, ideally leading to higher quality software. Project participants learning these practices will be able to apply their testing knowledge in future projects, which would support diffusing these practices in organizations.

Projects should visibly communicate their testing culture by providing a high degree of social transparency. Showing that the normative behavior in a project is to provide certain kinds of tests should help developers adopt these practices more easily. Testing culture can be picked up by new developers if they can observe the discussions surrounding changes, making it easier to understand the requirements of a project and the rationales behind them. This may be supported by clearly communicating what the testing status of a project is, e.g. by displaying a badge or by providing a project dashboard.

Similar to the study by Singer et al. [4], our results indicate that software developers value recognition by peers. Such effects can be applied systematically to improve development processes in organizations [26].

2) Software Engineering Research: Researchers may use our results as a starting point for more focused investigations in detail areas that we discovered. For example, *drive-by commits* are exemplary for social media — they are very small contributions that are easy to make — similar to a *Like* on Facebook. The usefulness of such contributions may be questionable, but their pervasive adoption cannot be denied. Researchers might want to investigate the conditions under which such contributions *can* be useful, and whether these can be created in closed software development organizations.

Our preliminary guidelines could be the first steps of a more exhaustive and validated *social testing framework*. This might support companies in creating their own tailored project guidelines that prescribe how testing culture is to be communicated in their projects. Companies struggling with high employee turnover might especially profit from knowing how new hires can learn about a project's testing culture most efficiently.

VIII. Limitations

Our research is a first, exploratory investigation into the effects that the characteristics of social coding sites like GitHub may have on testing practices. Therefore, we chose an approach based on Grounded Theory. While we achieved saturation in our interviews, it is likely that we did not reach all possible perspectives on GitHub use. While we sent out interview invitations to active but random users of GitHub, the final interview participants were all self-selected volunteers.

Similarly, the participants of our questionnaire were again chosen randomly, but ultimately were self-selected. The quantitative validation of our results therefore is again only applicable to the volunteering sub-population. The general population of GitHub might have different characteristics and opinions.

Apart from the questionnaire, we cannot provide quantifiable results. We cannot judge the strength or pervasiveness of any of the presented processes, mechanisms, or effects.

Finally, our results are not generalizable. We provide a view of testing on GitHub as seen by a self-selected population.

Yet, our research identified current challenges and solutions that are used in commercial and hobbyist open source software development. These can now be investigated in more detail with regard to their efficiency, effectiveness, and generalizability. Software developers and software development organizations may use them as inspiration for their own measures that they would need to validate independently.

IX. Conclusions & Outlook

When hosting a project on a social coding site such as GitHub, project owners interact with external contributors with varying knowledge and values regarding testing. Communicating a project's testing culture to such a population is an important, yet difficult task.

Our study reports on the influences of GitHub's peculiarities on testing practices — especially its high degree of social transparency, low barriers, and high degrees of integration and centralization. We present insights about the contribution process on GitHub and show how project owners assess pull requests with regard to testing. We found several testing-related challenges that members of GitHub face — and the strategies they have developed to cope with them, helping them create a shared understanding of a project's testing culture.

Developers on social coding sites can use our findings to gain insights into issues contributors may face and the strategies that can be used to handle them. Software development organizations may take our contributions as inspiration for basing their own testing-related communication strategies on.

This research is an exploratory first step to gain an understanding of the testing norms, challenges, and strategies on social coding sites. It is a starting point for informed, in-depth investigations into several of the issues raised in our study.

References

[1] M. Storey, C. Treude, A. van Deursen, and L. Cheng, "The impact of social media on software engineering practices and tools," ACM, pp. 359–364, 2010.

[2] C. Parnin and C. Treude, "Measuring api documentation on the web," New York, NY, USA, pp. 25–30, 2011.

[3] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in github: transparency and collaboration in an open software repository," ACM, pp. 1277–1286, 2012.

[4] L. Singer, F. Figueira Filho, B. Cleary, C. Treude, M.-A. Storey, and K. Schneider, "Mutual Assessment in the Social Programmer Ecosystem: An Empirical Investigation of Developer Profile Aggregators," University of Victoria, Tech. Rep. Technical Report DCS-347-IR, 2012.

[5] H. C. Stuart, L. Dabbish, S. Kiesler, P. Kinnaird, and R. Kang, "Social transparency in networked information exchange: a theoretical framework," New York, NY, USA, pp. 451–460, 2012.

[6] N. Nagappan, E. Maximilien, T. Bhat, and L. Williams, "Realizing quality improvement through test driven development: results and experiences of four industrial teams," *Empirical Software Engineering*, vol. 13, pp. 289–302, 2008, 10.1007/s10664-008-9062-z.

[7] A. Strauss and J. Corbin, *Grounded Theory in Practice*. SAGE Publications, 1997.

[8] danah m. boyd and N. B. Ellison, "Social network sites: Definition, history, and scholarship," *Journal of Computer-Mediated Communication*, vol. 13, no. 1, pp. 210–230, 2007.

[9] M. Greiler, A. v. Deursen, and M.-A. Storey, "Test confessions: a study of testing practices for plug-in systems," Piscataway, NJ, USA, pp. 244–254, 2012.

[10] M. Michlmayr, F. Hunt, and D. Probert, "Quality Practices and Problems in Free Software Projects," pp. 24–28, 2005.

[11] A. Mockus, R. Fielding, and J. Herbsleb, "A case study of open source software development: the apache server," pp. 263 –272, 2000.

[12] A. Bonaccorsi and C. Rossi, "Comparing motivations of individual programmers and firms to take part in the open source movement: From community to business," *Knowledge, Technology and Policy*, vol. 18, pp. 40–64, 2006.

[13] S. Krishnamurthy, "On the intrinsic and extrinsic motivation of free/libre/open source (floss) developers," *Knowledge, Technology and Policy*, vol. 18, pp. 17–39, 2006.

[14] C. Bird, "Sociotechnical coordination and collaboration in open source software," pp. 568 –573, sept. 2011.

[15] K. Crowston, K. Wei, Q. Li, U. Y. Eseryel, and J. Howison, "Coordination of free/libre and open source software development," p. Paper 490, 2005.

[16] K. Crowston, K. Wei, Q. Li, and J. Howison, "Core and periphery in free/libre and open source software team communications," p. Paper 489, 2006.

[17] K. Crowston, Q. Li, K. Wei, U. Y. Eseryel, and J. Howison, "Self-organization of teams for free/libre open source software development," *Information and Software Technology*, vol. 49, no. 6, pp. 564 – 575, 2007.

[18] K. Crowston and B. Scozzi, "Bug fixing practices within free/libre open source software development teams," *Journal of Database Management*, vol. 19, no. 2, pp. 1 – 30, 2008.

[19] K. J. Stewart and S. Gosain, "The impact of ideology on effectiveness in open source software development teams," *MIS Quarterly*, vol. 30, pp. 291–314, 2006.

[20] B. Dagenais and M. P. Robillard, "Creating and evolving developer documentation: understanding the decisions of open source contributors," New York, NY, USA, pp. 127–136, 2010.

[21] S. K. Sowe, I. Stamelos, and L. Angelis, "Understanding knowledge sharing activities in free/open source software projects: An empirical study," *Journal of Systems and Software*, vol. 81, no. 3, pp. 431 – 446, 2008.

[22] A. Begel and T. Zimmermann, "Keeping up with your friends: function foo, library bar.dll, and work item 24," New York, NY, USA, pp. 20–23, 2010.

[23] C. Souza and D. Redmiles, "The awareness network: To whom should i display my actions? and, whose actions should i monitor?" in *ECSCW 2007*, L. Bannon, I. Wagner, C. Gutwin, R. Harper, and K. Schmidt, Eds. Springer London, 2007, pp. 99–117.

[24] J. Lave and E. Wenger, *Situated learning: Legitimate peripheral participation*. Cambridge University Press, 1991.

[25] E. M. Rogers, *Diffusion of Innovations*, 5th ed. Free Press, 2003.

[26] L. Singer, "Improving the Adoption of Software Engineering Practices Through Persuasive Interventions," Ph.D. dissertation, Gottfried Wilhelm Leibniz Universität Hannover, 2013.