Let's Talk About It: Evaluating Contributions through Discussion in GitHub

Jason Tsay, Laura Dabbish, James Herbsleb
School of Computer Science and Center for the Future of Work, Heinz College
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213 USA
{jtsay, dabbish, jdh}@cs.cmu.edu

ABSTRACT

Open source software projects often rely on code contributions from a wide variety of developers to extend the capabilities of their software. Project members evaluate these contributions and often engage in extended discussions to decide whether to integrate changes. These discussions have important implications for project management regarding new contributors and evolution of project requirements and direction. We present a study of how developers in open work environments evaluate and discuss pull requests, a primary method of contribution in GitHub, analyzing a sample of extended discussions around pull requests and interviews with GitHub developers. We found that developers raised issues around contributions over both the appropriateness of the problem that the submitter attempted to solve and the correctness of the implemented solution. Both core project members and third-party stakeholders discussed and sometimes implemented alternative solutions to address these issues. Different stakeholders also influenced the outcome of the evaluation by eliciting support from different communities such as dependent projects or even companies. We also found that evaluation outcomes may be more complex than simply acceptance or rejection. In some cases, although a submitter's contribution was rejected, the core team fulfilled the submitter's technical goals by implementing an alternative solution. We found that the level of a submitter's prior interaction on a project changed how politely developers discussed the contribution and the nature of proposed alternative solutions.

Categories and Subject Descriptors

D.2.6 [**Programming Environments**]: Integrated Environments; H.4.3 [**Communications Applications**].

General Terms

Management, Design, Human Factors.

Keywords

GitHub; transparency; open source; social computing; social media; contribution; discussion; evaluation

1. INTRODUCTION

Open software development environments enable submitting code to any project. This means many people with diverse expertise – the "long tail" of contributors– can add unique value to a project. At the same time, openness poses the formidable problem of evaluating these contributions to ensure their quality and to maintain technical integrity. When contributions are deemed unsuitable or threaten technical integrity, a negotiation between contributor and project members often ensues [16]. Our work examines how this negotiation unfolds in an open environment, where project members and contributors have little formal authority to rely on in exerting influence.

For traditional open source software projects, the contribution process is characterized as a meritocracy where "code is king" [19]. According to this view, the decision to accept a code contribution largely depends on technical merit, and review of submissions identifies technical defects [15]. Technical merit, of course, is a rich and complex notion, including not just correctness, but notions of scope, style, design choices, priorities, and inter-project dependencies. However, previous studies on open source software suggest that the evaluation of a code contribution is often a more nuanced process involving many factors other than technical merit [7, 13, 15]. For example, newcomer or non-member contributions are often summarily rejected for violating project norms [13]. Contribution acceptance is higher for submitters with existing relationships with core members of a project, controlling for a number of technical factors [23]. These previous results suggest that the contribution evaluation process is more socially charged than the 'code is king' mantra would imply.

Extended code contribution discussions reveal important social project dynamics and project members' values and mental processes as they articulate arguments for or against a particular change. While the average GitHub contribution generates little if any discussion and involve very few developers [9], cases of extended discussions occur as project members and the broader community work to understand the implications of the suggested change [15]. Changes to a project often have major impacts on other related projects with different and perhaps conflicting goals [5]. Previous work has found that more complex code contributions to projects (in terms of lines of code and number of files changed, and whether the change was previously defined) are more likely to spur longer discussions about the suggested changes [6, 15, 23]. There is also evidence that existing relationships with project members may influence the nature of these discussions, as recent work suggests lengthy discussions around newcomer contributions have a very different outcome than discussions around member contributions [23]. Although previous work has noted the prevalence of extended contribution discussions in response to complex contributions, we lack a detailed understanding of the form and content of these discussions.

A study of the content and form of comments in lengthy discussions in response to code contributions would extend our

FSE'14, November 16–21, 2014, Hong Kong, China
ACM 978-1-4503-3056-5/14/11
http://dx.doi.org/10.1145/2635868.2635882

understanding of the dynamics of collaboration in an open environment. The problems and issues identified in contributions reveal unique challenges of collaborating with a heterogeneous diffuse set of contributors with limited shared understandings and common goals. In an open setting, project members have little or no formal authority over contributors and vice versa [14]. The methods and means they use to attempt to increase their likelihood of acceptance may reveal what constitutes power in this setting. Given the open nature of the environment, there is no prescribed method by which contributions are resolved. What do the outcomes of these lengthy discussions look like?

By better understanding the nature of comments in long discussions around contributions to open software projects, we can inform policies and tools that enable software developers to better manage open projects. For example, if contributions frequently result in certain types of conflicts, then collaborative software development environments can incorporate specialized conflict management tools [25]. Notification mechanisms (e.g., watching, following, mailing list subscriptions) can be more precisely targeted to those whose interests are affected.

To understand extended contribution discussions, we studied interaction in GitHub (github.com), a successful example of an open development environment. On GitHub participants submit contributions to projects by sending what is known as a pull request [8]. We analyzed a set of pull requests generating extended discussions from a large sample of open source software projects on GitHub. Average-case pull requests typically generate little to no discussion [9] and therefore are not very informative about the reasons for acceptance or rejection. For this reason, we chose to focus on only pull requests with extended discussions, which often reveal the developers' reasoning process. Using a grounded approach we focused on the phenomenon of extended contribution discussions (rather than interactions around contributions more broadly) because these discussions are an important aspect of the open collaboration process and a key place where core and peripheral members negotiate around aspects of project evolution and direction [4]. We explored the kinds of issues core developers raised and the arguments they over both the appropriateness of the problem that submitters attempted to solve, and the correctness of the implemented solution in a submitted code contribution. Due to the open nature of the software projects, other stakeholders from outside the project observed and participated in these extended discussions, sometimes attempting to influence the outcome of the contribution through rallying support of the audience or leveraging project or company communities. We also found that non-member contributions were more likely to be rejected following a long discussion. However, although core teams rejected new submitter's contributions more often, they almost always satisfied the submitter's technical goal by implementing an alternative solution. Core members interacted more politely with new submitters and in cases of conflict, were more likely to implement alternative solutions for these newcomer submitters rather than simply suggest them. In the next sections we motivate our research questions based on previous research, describe our research setting and study methodology, present the results of our discussion analysis, and discuss the implications of our findings.

2. CONTRIBUTION AND DISCUSSION IN ONLINE WORK

Previous research on open contribution evaluation suggests a number of different ways discussions are used to resolve issues that arise around contributions. This work suggests that discussions around newcomer or non-member contributions may evolve differently than for members, with different consequences. We position our work in the literature of evaluating contributions in peer production communities such as Wikipedia. We also examine how developers in open source software projects ensure the technical correctness of contributions. When more information is available in open transparent environments such as GitHub, developers use this information when evaluating contributions. Informed by previous work, we develop research questions around how developers discuss contributions in open environments.

2.1 Deliberation in Online Communities

One can consider an open source software project as a type of online community. Online communities are centered around contributions from a wide variety of users. Successful online communities rely on members contributing their unique resources to the community, such as users uploading videos on YouTube or posting pictures or comments on reddit. Kraut and Resnick [12] analyze challenges that online communities face when trying to encourage contribution: matching users to contributions needed, making requests to members, using intrinsic and extrinsic motivators, and grouping users together. They review evidence showing that constant feedback to members, whether it be character levels in World of Warcraft or community comments in YouTube, motivates members to create more contributions. Similarly, combining contributions with social contact also encourages further contributions. For example, the GNOME software project encourages socialization through forums and get-together conferences [12].

The contribution evaluation process can have an important impact on contributor motivation particularly for new members. In a study of newcomer contribution on Wikipedia, Halfaker et al. found that reverts decreased motivation for newcomers. Reverts from experienced editors were the most demotivating [10], suggesting that certain interactions around contributions may have a particularly negative influence on motivation to contribute to a project. Bryant et al. [2] found that contribution acceptance is an important step in a newcomers socialization process. Newcomers learn the conventions and contribution rules of the Wikipedia community through observation (lurking) and direct mentoring from more experienced users.

However, conversation on Wikipedia is most often used to organize work, rather than to discuss issues or problems with contributions. Viegas et al. [24] found that in Wikipedia editors primarily used edit comments to coordinate article edits. Some research has looked in detail at how the content of communication influences participation, for example finding that politeness strategies in an opening thread on a discussion forum increases the likelihood of reply [3]. However, participation in forum-based online communities is typically focused on social support and information exchange, rather than collaboration around a shared project. Thus we may expect to see new types of conversational dynamics in contribution evaluation discussions where the contribution is the focus.

2.2 Contributions in Open Source Software

As open source software often relies on the contributions of a diverse group of software developers [5], members of software project teams must evaluate and discuss contributions to ensure the integrity of the software project.

Literature on the contribution process for open source software projects suggests that evaluating contributions, especially from

unknown developers, is a complex social process. Krogh et al. [13] found in their study of the contribution process in the Freenet open source project that successful newcomers must follow "joining scripts" before submitting a contribution. These joining scripts involve participating in prior activity such as lurking on the project's mailing list, participating in technical discussions, and reporting bugs. They also found differences in the tone of discussion between developers who were invited to join the project versus developers who were not. For example, the detail and specificity of feedback given was much more general for non-joiners. Ducheneaut [7] noted that developers looking to make successful contributions to the Python project needed to undergo a progressive socialization process. Core members on a project would vet contributions to ensure the code changes were technically sound. Successful socialization allowed potential submitters to learn project norms and to identify members of the core project team. In order to successfully start the contribution evaluation process, a submitting developer needed to "recruit" core members of the project as a network of "allies".

When evaluating code contributions for technical correctness, core project members often use a peer review process. Rigby et al. [17] found in their examination of different peer review processes in the Apache server open source project that early and frequent reviews of small contributions from the core team were effective in finding defects in contributions. In particular, the usage of the project mailing list allowed for self-selection of expert core members and a more open discussion between members. Ko and Chilana [11] found that discussions around bug reports established scope, proposed ideas, identified design dimensions, defended claims with rationale, moderated the process, and finally made a decision. The most powerful factors in decision-making around a bug report were the participant's authority (developers over users) and actions taken (writing a patch).

2.3 Transparent Work Environments

While evaluating contributions is a key process in all popular open source projects, different environments provide different tools and mechanisms for participating in discussions and negotiations about them. The setting for our study is GitHub, often cited as an example of a transparent environment [4]. Transparent environments incorporate social media features that allow developers to utilize a much greater range of information when making and evaluating contributions due to the visibility of work across the entire community. Previous qualitative research on GitHub by Dabbish et al. [6] showed that developers use this information available in GitHub through social media features to make a variety of subtle inferences about other developers and projects.

Project managers, especially those in popular projects that received many contributions (pull requests) per day, used information available to them via transparency in order to assist in the evaluation process by making inferences about the quality of code contributions and submitter competence. At times, project managers needed to communicate directly with submitters about code contribution. Most often, this was done to negotiate additional changes. Other times, core members needed multiple rounds of discussion with the submitter in order to create a shared understanding of what the submitter's contribution was attempting to do and what the current direction was at the time of submission. Core members also used inline interaction with specific lines of code in order to address specific conflicts or to ensure conformity with project style norms. The transparent nature of GitHub also led developers to become acutely aware that their work actions had an audience. Audience pressures led developers to change their behavior. For example, developers spent some time managing face by being careful not to offend developers by publically rejecting long time contributors or not following someone who followed them.

Marlow et al. [15] found that when GitHub developers engage in information-seeking behaviors, they used information made visible by the environment in order to form impressions of users and projects. Developers would engage in information-seeking behaviors after and during interactions with other developers on GitHub. For example, developers looked to another developer's previous work in order to better understand their coding abilities. When evaluating a developer's contributions, project managers would look to the submitter's other projects in order to better understand how much assistance or extra effort the submitter would require in order to accept their contribution. Core members would also often account for uncertainty when evaluating contributions. Uncertain changes required back-and-forth discussion between the submitter and core members in order to explain why the contribution could not be automatically accepted and negotiate the outcome. In these cases, project managers would make use of information about the code contribution and the submitter, to decide how accommodating towards the submitter they should be. For example, a project manager may weigh the cost of fixing a contribution against the benefit of recruiting a new member to the project.

Tsay et al. [23] found in a study of contributions in GitHub that a number of social and technical factors influenced the probability of a contribution being accepted. When contributions were highly discussed, contributions were much less likely to be accepted. However, the prior interaction of a submitter on the project moderated the negative association of high discussion on a contribution and acceptance. The submitter's prior interaction on a project also had a positive association with acceptance.

The literature on deliberation in online communities suggest that members engage in discussion to both encourage and evaluate contributions to the community [6, 12, 13]. Open source software projects, needing to ensure the technical integrity of code contributions, engage in complicated social processes and peer technical reviews [15]. Often, developers would also engage in discussion in order to socialize themselves when joining a project [6, 11]. Transparent work environments such as GitHub have developers using information made visible due to transparency to make inferences about projects and other developers when evaluating contributions [5, 13]. However, we still know relatively little about the kinds of issues that arise, and the nature of discussions developers have when evaluating contributions.

2.4 Development of Research Questions

Our examination of literature on discussion around contributions in online environments suggests a number of research questions to advance our knowledge of how software developers discuss contributions in open environments.

From online communities such as Wikipedia [24], we see that editors engage in discussion over conflicts in article direction. In open source software, developers discuss problems in bug reports, making and justifying arguments when discussing the design of a solution [11]. For more uncertain changes in GitHub, core members engage in back-and-forth discussion to justify the value of the contribution [15]. However, it is not well understood what issues around open source software development need to be worked out in these discussions. By better understanding the issues, arguments, and criteria raised in contribution discussion,

we can identify challenges in collaborating with a diffuse set of contributors in an open environment. This leads us to our first research question:

(1) *What are the different kinds of issues raised around code contributions?*

In online communities, members use intrinsic and extrinsic motivators when making requests to encourage compliance [12]. One tactic identified in open source software is that submitters recruit core members to assist in the evaluation process [7]. In this way, submitters are able to influence the outcome of the evaluation process. What is not well understood is the full range of methods that different stakeholders may use to influence the evaluation process. By better understanding the decision-making process and how influence is brought to bear, we also gain insight into what motivates software developers to accept changes. The influence tactics used in an environment where there is little formal authority [14] also reveal what constitutes power in open collaboration. This leads us to our second research question:

(2) *How do participants try to influence the decision process in code contributions?*

We find in online communities that the outcome of a contribution evaluation may be farther-reaching than simply whether the contribution itself is accepted. For example, the outcome of a contribution and the identity of the evaluating editor in Wikipedia has an impact on the motivation of a new editor to contribute again [10]. In open source software environments such as GitHub, project managers may decide to accept less desirable code contributions in order to recruit new members [15]. We investigate the following research question in order to better understand the less obvious impacts of outcomes of code contribution evaluations:

(3) *What are the different outcomes for proposed code contributions?*

In online communities, new submitters tend to interact differently. In Wikipedia, new editors tend to make peripheral, specific edits to articles [2] and tend to be sensitive to reverts from experienced editors [10]. In open source software, new contributors tend to follow "joining scripts" before making successful contributions [13]. Similarly, a submitter's level of prior experience on a project has an association with contribution acceptance and seems to moderate the negative effect of discussion [23]. This suggests that a submitter's prior experience (or lack of) changes the nature of discussion around code contributions and may influence the outcome of the evaluation. By understanding how a submitter's prior experience impacts discussion, we may better understand how projects manage both new and experienced contributors. This leads us to our fourth research question:

(4) *Is discussion different when the submitter has prior experience with a project?*

3. METHOD

To answer our research questions, we created and analyzed a dataset of both interview data and contribution discussions from the social open source software hosting site GitHub [8]. The interviews allowed us to investigate the practices of a relatively broad sample of developers, while the content analysis of entire collections of comments for specific pull requests allowed us to analyze complete exchanges and their outcomes in some depth. In this section, we present descriptions of the GitHub setting, our data collection methods, and analysis technique.

3.1 Research Setting

GitHub [8] is a software project-hosting site started in 2008 that brands itself "Social Coding" that is home to over ten million repositories [1]. Some of the more popular open source software projects that GitHub hosts include *Ruby on Rails* and *jQuery*. We selected GitHub as our research setting because it is an open environment where public software repositories are made available for anyone to participate. In GitHub, one of the main methods for contributing to a software project is by sending a "pull request". First, potential project contributors "fork" or make a personal copy of the target project where they can make changes to, add, or alter functionality. This potential contributor can then request that code changes in their fork be merged into the project's main repository. This can be accomplished by creating a "pull request*"* that contains the code changes that the submitter would like integrated into the original project. Core members in the original project have several options to "close" the pull request, including accepting the offered contribution and merging it into the project code base or rejecting the contribution. Of course, project managers may also ignore the contribution, leaving the pull request "open".

Core project members and other interested users can discuss the contribution by creating comments on the pull request page, to suggest improvements or ask questions about the code change. There are two different methods for commenting on a pull request: general comments about the contribution as a whole (see Figure 1), or code-level inline comments for specific lines in the code contribution (see Figure 2).

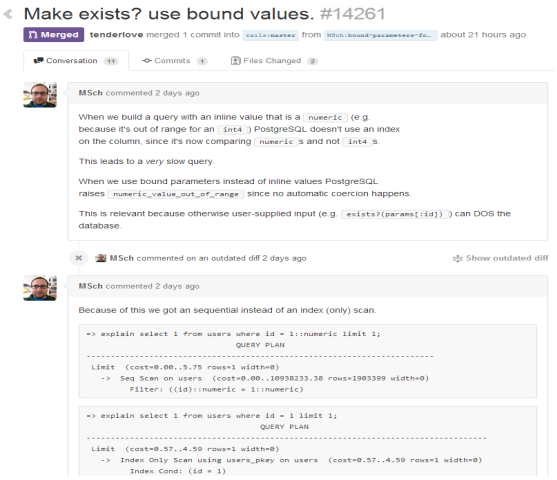

Figure 1. Example of pull request and discussion

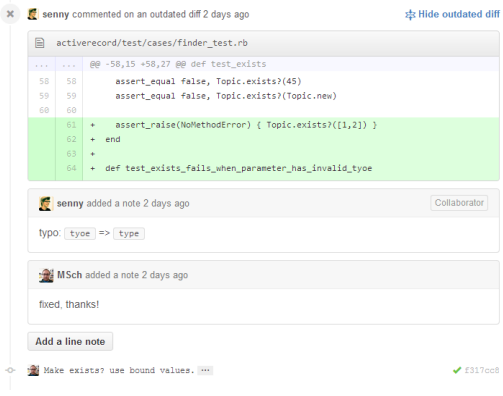

Figure 2. Example of inline code comment for pull request

3.2 Data Collection

Our dataset consists of both a set of 423 comments from 115 developers, embedded in extended pull request discussions and interviews with 47 users of GitHub.

3.2.1 Highly Discussed Pull Request Sample

From a larger dataset of 659,501 pull requests across 12,482 GitHub projects [23], we created a sample of highly discussed pull requests (see Table 1). We defined "highly discussed" as pull requests where the number of comments is one standard deviation (6.7) higher than the mean (2.6) in the dataset, filtering out all pull requests with less than 9 comments in the discussion. For each pull request, we include both discussion comments on the pull request itself and code-level inline comments. From this reduced sample, we randomly selected 20 highly discussed pull requests from 20 different software projects. From these 20 pull requests[1], a total of 423 comments from 115 developers were analyzed. As we reached theoretical saturation, we drew no more pull requests from our sample. We constrained our sample to ensure that it included both accepted and rejected pull requests, as well as submitters with varying levels of prior interaction with the project (see Table 2).

As our research questions are concerned with the outcomes of contributions, we wanted a roughly equal representation of both accepted and rejected pull requests. As we were also investigating how a submitter's prior experience on a project changes the discussion around a contribution, we also wanted a roughly equal distribution of new submitters (no prior interaction) and experienced submitters (see Table 2).

Table 1. Description of pull request sample

Number of pull requests	20
Total number of comments	423
Mean number of comments	21.1
Total number of participants	115
Mean number of participants	5.75

Table 2. Distribution of pull request sample

	Yes	**No**
Pull Request Accepted?	9	11
Submitter Has Prior Interaction on Project?	11	9

3.2.2 Interview Data

To supplement our sample of extended discussions, we also conducted a series of semi-structured interviews with 47 GitHub users [6]. Our goal in these interviews was to document and understand in more detail the different ways GitHub functionality was used by our participants, including how pull requests were created and managed. We solicited participants via email and conducted our interviews in person or via phone or Skype. Remote participants shared their screen during the interview using Adobe Connect so users could demonstrate their activities on the site. Participants were asked to walk through their last session on GitHub, describing how they interpreted information displayed on the site as they reviewed earlier work activities. For this study, we used responses to that portion of the interview where participants were asked to describe their last pull request sent to a project and to describe the last pull request received for their own project. None of the pull requests mentioned in interviews were included in our sample of pull request discussions. Interviews lasted approximately 45 minutes to one hour overall. These interviews were then transcribed verbatim to support further analysis.

[1] http://jsntsay.com/work/FSE2014.html

3.3 Data Analysis

We applied a grounded theory approach to analyze how developers evaluate contributions in transparent environments in our sample of pull request discussions [21]. We first identified instances of how developers evaluated code contributions in the comments of five pull request discussions. For each instance analyzed, we identified the participants involved, information made available by GitHub that is used by developers, the type of comment, what portion of the code contribution is referred to, and the higher-level goal of the participant in regards to the contribution. We then conducted open coding on these examples, grouping examples into categories that were conceptually similar. This process revealed different categories of interaction between different types of participants for a code contribution. We used this first set of categories to code the remaining pull request discussions, revealing additional categories. We used an iterative process until the discussions no longer revealed new interactions not captured in our existing set of categories (theoretical saturation). During this process, we also identified similar interactions in our interview data and used these examples to supplement our pull request discussion examples.

4. RESULTS

Our analysis found that both core and peripheral developers in a project engaged in discussion in order to resolve issues around both the problem that the contribution is attempting to solve and the solution that the contribution implements. Different stakeholders such as third parties and audience members sometimes attempted to influence the outcome of discussion. We found different outcomes for contributions and discussions around them. We also found that the submitter's level of prior interaction on the project changed the discussion around the code contribution.

4.1 Issues Raised Around Code Contributions

(1) *What are the different kinds of issues raised around code contributions?*

Contributions to projects in the form of pull requests sometimes generated issues that the submitter and the core members must resolve through discussion. We saw that core members raised different issues over the appropriateness of the problem that the contribution was attempting to solve. We also saw that developers discussed how to optimize the solution that the contribution implements with various levels of involvement. At times, core members in the project also disagreed amongst each other over these issues.

4.1.1 Disapprove of the Problem Being Solved

One main issue that developers discussed was whether the problem that the code contribution was trying to solve is appropriate. Core members would sometimes discuss whether the pull request belonged in their project while other times would ask contributors to prove the value of their contribution through explicit use cases.

4.1.1.1 Project Appropriateness

Core members questioned whether the submitter was using the project in the intended manner. Sometimes, contributions offered by submitters using the project inappropriately would attempt to implement features that were not in the intended scope of what the project was meant to do (P2, P7, P14, P15, P16, P19, P20). In

these cases, core members offered alternative solutions to the submitters outside of their own project. One example had the project owner offer to assist the submitter in learning how to use the project correctly offline, in a local hackathon (P15). If the contribution was outside the scope of the project, core members sometimes suggested that the contribution actually be made to an upstream or downstream project (P20).

To prevent submitters from wasting time on inappropriate contributions, core members expected submitters to propose their contributions before implementation to get feedback on its appropriateness (P7). In cases when core members inadvertently accepted inappropriate earlier contributions into the project, later contributions would be necessary to revert the inappropriate change (P20), further increasing the time and effort wasted for both submitters and core members.

"The idea of proposals & issues before-hand is to see the likelihood of something getting merged, so you don't feel you've wasted all your time if it doesn't." (P7)

Some GitHub developers explained in interviews that submitters would sometimes inadvertently solve inappropriate problems because the project would move in a different direction unknown to the submitter. For example, a core developer's planned changes to a project made a submitter's contribution obsolete.

"So, yeah. Not sure what's gonna happen with this off the top of my head, if it's gonna get landed or-- I mean because some of the things that we're doing with this refactor of the master branch make this whole thing a little unnecessary now."

4.1.1.2 Value Proposition Request

In order to explore whether the contribution truly had value for the project, core members asked submitters to provide specific use cases or test cases (P1, P2, P3, P4, P13, P14, P20). Core members used this requirement as a way to confirm that the specific problem submitters were trying to solve in their contribution was appropriate. In some discussions, core members refused to continue the evaluation process until use cases were presented (P1, P3, P4, P14).

In response, submitters offered use cases or test cases to demonstrate the problem their contribution solved. For the contributions in this category, submitters that provided code examples or references to downstream projects (P2) tended to successfully prove the appropriateness of their problem and also tended to have their contributions accepted. Occasionally, third parties from the audience would also jump in, offering their own use cases when core members asked for them (P13). When submitters were unable to satisfactorily demonstrate use cases that their contribution solves, the contribution tended to be rejected. In some cases, core member simply closed the pull request until a test case was provided (P1). In one case, the submitter, uncertain about their own use case due to discussion, decided to close their own pull request until a better use case was presented (P14).

"I think it may be better to close the pull and the associated issue unless I'm overlooking a real use case… I'm going to close this for now, if someone comes up with a good use case we can reopen." (P14)

4.1.2 Disapprove of the Solution

When core members and third parties from the audience questioned the solution that the pull request implemented, developers offered a gradient of responses to questionable solutions from passively questioning the submitter's approach to actively suggesting alternative solutions to offering their own solutions to the problem.

4.1.2.1 Question Solution Approach

Core members raised objections to the way the submitter chose to implement the solution in the contribution. Most often, developers raised questions about the submitter's approach to implementing the solution in the pull request (P2, P4, P8, P9, P10, P11, P17, P18, P20). In some cases, core members asked about design decisions that the submitter made when implementing the pull request (P2, P4, P8, P10, P18, P20). These decisions ranged from the forming of dependencies (P10, P20) to more elegant code (P4) to even best commenting practices (P8). Other cases had core members act as testers for the code change, reporting bugs with the pull request (P11, P17). In one case, the core member actually reported the bug after the pull request was accepted (P17).

"Wow. Don't you think you're going a little bit overboard with this many comments? Or is this just for my benefit when checking on your code?" (P8)

4.1.2.2 Suggest Alternative Solutions

Some core members and third parties from the audience took a more active approach when the contribution's implemented solution was suspect and suggested alternative solutions to the submitter's implemented solution, often with the expectation that the submitter would implement the suggestion (P2, P7, P8, P9, P11, P15, P19). Many of the suggestions given were technical in nature, suggesting ways to improve the code through optimization (P9) or better practices (P11, P15) or avoiding bugs (P7, P8). Others were more stylistic in nature, suggesting changes to conform to best practices or project norms (P19).

"I would suggest having an array of possible node locations and loop through them in order using fileExists to determine if it's available." (P15)

Submitters sometimes followed and sometimes ignored suggested alternative solutions. In one pull request, a core member made a suggestion for an alternative solution that the submitter accepted and attempted to implement. Not being able to implement the suggestion, the submitter decided to leave the pull request as-is (P11). In some cases, the submitter actively rejected the suggested alternative solution. In one case, a third party developer from the audience suggested an alternative solution that the submitter, a core member, rejected with an explanation why (P9). With these two examples, regardless of the outcome, submitters addressed suggestions from the core or third parties. In one case, a new submitter even preemptively addressed an obvious alternative solution, explaining why it would not be appropriate for the particular problem that the contribution was trying to solve (P2).

"You might ask: "why don't you install the suggested rb-inotify gem to avoid getting that [...] warning?" The reason is that such a task can only be performed by the end user who uses my scripts; I have no control over their machines" (P2)

4.1.2.3 Advertise Own Solution

As a response to issues in how the solution in the contribution was implemented, some core members or third party developers from the audience took the initiative to implement their own alternative solutions to the problem presented in the contribution and then advertise their own solution in the pull request discussion (P2, P3, P13, P15). The actual form of the alternative solution varied widely from case to case. Interested third party developers from the audience gave examples of solutions to similar problems that were implemented in outside projects that the developers

previously worked on and provided hyperlinks to that project (P2, P3). One third-party developer from the audience, in response to problems in the implemented solution, made suggestions for an alternative solution and then implemented the solution in another pull request (P3). This case created a competing solution to the same problem that the original contribution attempts to solve. In another case, a core member sent a pull request to the submitter's personal fork of the project that made code changes to the contribution, effectively making a contribution on a contribution (P13).

"@[submitter] I sent you a PR([link to pull request]) that accounts for once in the callback, avoiding a potential infinite loop. Test included too." (P13)

4.1.3 Disagreement among the Core

In almost a third of cases in our sample of highly discussed pull requests, core members disagreed amongst themselves in regards to the best way to approach a problem or what is the best possible solution for a contribution (P2, P3, P6, P8, P10, P11, P12). In these cases, core members often showed deference to more senior core members, often project owners or project creators (P2, P3). On the other hand, more senior core members used the opportunity to instruct or even admonish other core members (P2, P3). In some cases, senior core members even handed down edicts to what the project will do about a particular problem.

"@[not-as-core developer] The point is that we need to allow this kind of integration (that's part of the interoperability we try to promote)." (P3)

Core members, when disagreeing with each other, used various techniques to hedge their arguments. In many cases, disagreeing developers used humor and emoticons to soften their arguments (P2, P8, P10).

"Hey guys, sorry I'm a bit late but I don't feel comfortable with writing what's not diagnostic to me to STDERR. [...] We're losing control guys!!!!!! :P" (P2)

4.2 Methods of Influencing the Decision Process for Code Contributions

(2) *How do participants try to influence the decision process in code contributions?*

Various stakeholders involved in the code contribution employed different methods to influence the outcome of the contentious pull request. Third-party stakeholders in the audience at times applied pressure to core members to accept code contributions.

4.2.1 Audience Pressures

Third party developers in the audience held stakes in particular code contributions, often needing a particular change for their own usage. These interested audience members applied pressure to core members in order to influence their evaluation decision. Developers in the audience were able to pressure core members through rallying support from other developers and projects or companies.

4.2.1.1 Community Support

Outside developers in the audience with a stake in a code change commonly demonstrated support for a particular contribution by making comments in pull request discussions indicating that they needed the change (P1, P2, P3, P5, P7, P9, P13, P14, P16, P19). Most commonly, audience members indicated their support in the form of a "+1" or a "+1" emoticon (P2, P5, P13, P16).

"@[submitter] +1. It's very convenient for setting off one time operations that need to respond once to a recurring event, such as a set up operation." (P13)

Other than simply indicating support, audience members also commented that they were experiencing the same problem as what the code contribution fixes, increasing the perceived number of users that needed the change (P1, P7). In some cases, core members also indicated their support for a particular code change to other core members (P2, P3, P14, P19).

"Just to confirm that this issue still exists in master… The fix in the pull request works for me. Please consider merging." (P1)

Interviewed GitHub members explained that when they perceived that their community needed a feature through feedback, they were motivated to implement those features.

"I get feedback from those people and kind of think about and think, oh gosh, it looks like what they really need is this feature and this will work for them and I'll do the design."

Other interviewees complained about such practices, citing the noise that such community support brought when trying to discuss issues around code contributions.

"I mean it's kind of difficult to have a productive conversation about something like that when you get a million people coming in and just saying plus one, plus one, plus one, plus one"

4.2.1.2 Project and Company Support

In some cases, rather than simply indicating a need for the change, third party developers in the audience cited their own projects or companies that would benefit from the contribution in question (P3, P6, P13, P14, P16, P18). In these cases, developers intensified their stake in the code contribution, demonstrating that other projects or even companies were relying on the change to be accepted.

"@[core member] if you are still interested in finding a solution for this problem i can give you any details you need just ask. I'm very interested in solving this because we are investing a lot on [project] in my company but every single of our applications uses saml for authentication [sic]." (P3)

Developers also seemingly leveraged their own user base in order to exert influence on the contribution decision process. In one example, the submitter mentioned that the contribution was actually meant to solve a problem on behalf of one of the submitter's users (P3). Another case had the submitter mention that many users of the project switched to the submitter's fork of the project in order to avoid a particular bug that the contribution also fixes (P16).

"Several people have started using this fork in order to get around the issues reported in [issue link]." (P16)

4.2.2 Alerting the Core

In order to engage particular core members in the contribution discussion, both the submitter and core members made use of the @mention feature in GitHub, which notifies specific developers who are mentioned during discussion. (For example, @octocat sends an email notification to the developer with username "octocat".) Developers used the @mention feature in order to alert core members who are key to the evaluation process for the contribution in question (P2, P3, P13, P14, P16, P18, P19). Submitters or other core members @mentioned core members in order to start the code review process (P13, P16). Occasionally, the reverse also occurred, with core members @mentioning the submitter in order to continue the review process (P14). Often,

core members @mentioned other core members in order to solicit feedback from more qualified core developers (P2, P3, P18 [not an @mention but still a solicitation of feedback from fellow core members]). In one case, a core member alerted the rest of the core team before merging a code contribution in order to give other core developers an opportunity to comment.

"I think these two sketches look good, anyone see any issues with merging?" (P19)

In some cases, third party developers also @mentioned core members in order to attempt to influence their decision regarding the contribution (P13)

"+1 @[core member] please re-open this for consideration. .once does not provide the same functionality" (P13)

4.2.2.1 Submitter Asks Core About Evaluation Status

After periods of inactivity in the discussion around a contribution, submitters often asked the core team about the status of the evaluation process for the pull request (P6, P13, P16). The periods of silence before a submitter asked about status ranged from 18 days (P13) to 2 months (P16). Inactivity, as discussed in a later section, caused developers to fear that their contributions were ignored by the core team.

"Anything I can do to get this merged? @[core member] @[core member] ?" (P13)

4.3 Outcomes for Proposed Code Contributions

(3) *What are the different outcomes for proposed code contributions?*

Based on prior work on the factors that influence what pull requests are accepted in GitHub [23], highly discussed contributions tended to be rejected while submitters prior interaction on a project tended to have their contributions accepted. In this work, we have an opportunity to examine in detail what types of discussions result in rejected or accepted pull requests. With the different issues raised around code contributions, we also saw different methods of resolution and different behaviors after a pull request is resolved.

4.3.1 Rejection and Meeting Technical Goals

One finding from the analysis of discussions is that while many of the highly discussed pull requests we examined were rejected, the core team would often still meet the underlying technical goal of the submitter (P3, P7, P13, P15). For example, in a few contributions, the core team realized during the discussion around the contributions that the underlying problem that the submitter was attempting to solve was much more complicated than originally thought. After discussing the contributions, the core team decided to implement their own, more complete, solution to the original problem (P3, P13). In this way, although the submitter did not have their contribution accepted, the core team fulfilled the submitter's technical goals. In one case, the submitter had submitted a malformed pull request, leading to its rejection. Rather than resubmitting the contribution, the submitter instead asked a core member to implement the bugfix. In this case, the submitter was more interested in meeting personal technical goals than having "credit" for having an accepted pull request (P7).

"So I'm going to let [core member] decide what he want to do with it. It's an easy search&replace action, so it doesn't have to be this PR." (P7)

4.3.2 Contribution Outcomes

When submitters or third-party stakeholders exerted influence through audience pressures, the pull request we examined were no more likely to be accepted by core members (4 rejected and 7 accepted). However, with the exception of one pull request (P1), whenever the audience influenced the outcome, the technical goal of the submitter was met, either through the contribution being accepted or the core team implementing their own solution to the problem in the contribution (P2, P3, P5, P6, P7, P9, P13, P16, P18, P19).

In cases where the problem that the contribution was attempting to solve was suspect, especially when the project usage or scope was inappropriate, the contribution tended to be rejected (P1, P3, P4, P7, P13, P14, P15, P20). In the two exceptions (P2, P16), where the problem the contribution solved was suspect yet the pull request was accepted, the core team disagreed amongst each other, engaging in extended discussions about the contribution.

4.3.3 Future Contributions Advertised

After contributions were resolved, submitters often advertised future changes in the discussion (P2, P4, P7, P11, P18, P19). Even if the contribution was rejected, submitters sometimes offered suggestions on similar changes in the same direction as the offered contribution (P4, P7. P11).

"I'm closing this for now, as this needs more testing. I would also like to investigate whether we can support multi-monitor configurations better than today." (P11)

In changes that were accepted, some submitters indicated future changes that were incoming (P18, P19).

"Let us know about syntax, formatting etc. on these 4. We 50 more in the pipe passing internal peer review. […] We're upping our schedule to get more pages. Out we have a ton in the works but need to sign off on them internally before handing them over. " (P18)

4.4 Submitter's Prior Experience

(4) *How does a submitter's prior experience with a project change the discussion?*

Prior work on factors influencing pull request acceptance in GitHub [23] found that a submitter's prior interaction had an influence both on whether pull requests and highly discussed pull requests were accepted. In this work, we were able to examine how a submitter's experience changed the nature of discussions around their contributions.

4.4.1 Core Thanking New Submitters

When submitters were new to the project, core members were sure to politely engage with the new submitter regarding their contribution. For new submitters, core members thanked the submitter for their contribution as their first comment (P2, P5, P6, P11, P12, P13, P18, P19). In other cases, core members apologized to new submitters for delays in responding (P1, P5, P6, P10, P12, P13, P18, P20). Often, developers in GitHub interpreted delays in response as the core project team ignoring a contribution and use this information as a signal for poor project management. Often, the first comment to a new submitter combined the two, both thanking a new submitter for their contribution and apologizing for a delay in response at the same time.

"Looks impressing. Since I'm a bit busy with some other stuff I'll made a review in a week or something. Please be patient. And thank you for contribution :)" (P5)

Interviewed GitHub developers were aware of the value of being courteous in regards to accepting pull requests.

"I mean if there's a problem with the library you don't want to rush in and say, "Your library sucks, and it's wrong in the following ways and I'll fix it for you. You need to merge it," or whatever. It really comes off badly [...]. But if you come at it from another direction and say, "This is a great library. Thanks for providing it. I do have one or two little changes that I'd like to make. I think it'd help the library as a whole. What do you think?" That generally comes off much, much better."

Interviewees also explained that they would be polite to new submitters to try to encourage contributions.

"For smaller things, does it help people to contribute? So I think that this is kind of entirely an issue of how do you handle it [...]. Not to say that you're always squashing their ego or putting them down when you're making these changes for them. I think that you can say, "Hey, thank you for the pull request. There were some issues here, here, here that I fixed up and then I merged it. In the future try and make sure that you do this. Thanks again, though, for the code." Usually people respond pretty positively to that."

4.4.2 Alternative Solutions for New Submitters

When the submitted contribution's implemented solution was suspect, depending on the level of the submitter's prior interaction on the project, core members and third parties had different responses when offering alternative solutions. In general, regardless of the submitter's experience on the project, other developers questioned the approach of the contribution's implemented solution. However when discussing alternative solutions to the contribution, the prior experience of the submitter seemed to change how developers offered their alternative solutions. Submitters with experience on the project tended to receive suggestions on alternative solutions to solve the contribution's problem (P7, P8, P9, P11, P15) while new submitters to a project tended to receive implemented alternative solutions from core members and third party developers (P2, P3). In both pull requests, multiple alternative solutions were advertised, ranging from competing contributions (P3) to similar solutions in other projects (P2, P3).

5. DISCUSSION

In our observations of the interview and pull request discussion data, we found that developers were very aware of the different stakeholders when discussing contributions. Developers also had multiple methods of influencing the evaluation process, including influencing power relationships in the project. We also found that core members and submitters defined and evolved project requirements during discussions around code contributions.

5.1 Stakeholders Influencing the Outcome

One of the side effects of open collaboration is that the environment allows for third party developers to participate in discussions around evaluating contributions. In open environments, a project's dependencies are not fully known to the core members. Any developer can independently use any library. Notification mechanisms, such as GitHub alerts, make developers in the audience aware of important changes that may affect them.

While prior work on GitHub has suggested that the presence of a perceived audience itself pressures developers into behaving differently [6], our findings suggest that the audience takes on a much more active role when evaluating contributions. Similar to developers overhearing discuss ions in collocated software teams [22], we found in our sample of pull requests with extended discussions that developers in the audience would often jump into discussions where they may have stakes in the outcome. Gousios et al. [9] found in their sample of pull requests that discussion participants who have never committed to the repository are rare. We found, however, that extended discussions tended to draw developers who were not directly related to the pull request, i.e., were neither the submitter nor core members. Most of these third party developers made some peripheral contribution to the project at some point.

The ability of third party developers to independently join the discussion around any contribution may influence how core members and submitters evaluate and discuss contributions. Submitters received suggestions from both core members and third party developers from the audience and would often need to justify their design decisions. In some cases, submitter's solutions even competed with alternative contributions from third party developers that solved the same problem as the submitted contribution. The extra negotiation required due to suggestions from the audience may raise the cost for core members to evaluate a pull request, reducing its chances of acceptance [23]. At the same time, this exploration of alternative solutions by the audience seems to be a form of decentralized experimentation. So while core developers may be less willing to make risky experimental code changes while being watched, third parties from the audience may be willing to take on the risk.

Software development environments with pervasive notification mechanisms such as GitHub allow developers the affordance of staying aware of projects where they may be stakeholders but not necessarily core members. This awareness has the side effect of creating an audience that may actively attempt to influence the development of a software project through participating in discussions or developing experimental code changes. Future research should explore how notification mechanisms enable developers to be actionably aware of projects they may have stakes in. A better understanding of how developers act on awareness notifications would inform the design of tools that better notify developers when to participate in relevant discussions in dependent projects and allow core members to effectively manage experimentation from third party stakeholders.

5.2 Power Relationships in Evaluating Contributions

Discussions around contributions had three types of participants: submitters, project core members, and third party audience members. These three groups of developers appeared to have implicit power relationships.

The closer the developer was to the project's core, the more influence the developer seemed to wield. For example, a third party developer's suggestion had much less weight to the submitter than one from a core member. Core members had the ultimate power to accept or reject a code contribution due to their commit access. The degree of influence also varied within the core, with certain core members showing deference to more senior core members such as project owners or veteran contributors. Submitters, having implemented a solution, demonstrated investment in the project. Third-party stakeholders had not demonstrated such investment.

To help determine power relationships in a project, developers used information present in the environment to make inferences on the expertise of other developers [15]. Core members and submitters may attribute less influence to the comments of a third party due to inferences made using cues in the environment. For

example, a third party developer with no connection to the project may be seen as a certain "type of person" who only reports problems but does not actually contribute code [15] and therefore may have less of a stake in the technical discussion about the contribution. The submitter's prior experience on the project was also used as information to infer the submitter's expertise. Prior experience may be an indicator of the degree of socialization a developer has undergone for the project [7]. Socialized developers, possessing knowledge of the core team and project-specific norms, may be less likely to create risky contributions or contributions of uncertain value.

While third party and submitting developers may wield less power than core members during the contribution evaluation process, these developers were able to leverage their own communities to influence the core team on a project. We saw that developers would cite their own projects and companies in order to intensify their perceived stake in the code contribution, perhaps increasing their influence on a change through pressure [12]. Leveraging user bases in this way to influence the core was often effective because core members understood that their authority is closely tied to keeping users satisfied [14]. In some cases, we saw that stakeholder communities would actually cause a submitter to create the contribution in the first place. For example, if a user was experiencing a bug in a certain project, the project owner implemented and submitted a bug fix to an upstream project [6]. This suggests a chain of influence across the upstream and downstream dependencies in software projects. The pressure to contribute to an upstream project may have benefits to the technical integrity of both projects due to ensuring that a code change resides in the most appropriate location in terms of architecture. For example, if a bug goes unfixed in an upstream project, multiple downstream projects may all have to implement the same workaround or bugfix.

How these power relationships between open source developers as well as the incentives and decision rights that are present support good decision-making in terms of evaluating code contributions is not well understood. Future research should investigate these relationships in more detail, in order to determine what factors allow developers to wield more influence than others when making evaluation decisions. Environments that make these factors such as expertise visible or allow for different notification capabilities may have an impact on these power relationships and the outcome of code contribution evaluations.

5.3 Developing Software Requirements Through Discussion

Core members and third party developers from the audience often raised issues around a contribution, either about the appropriateness of the problem solved in the pull request or the correctness of the implemented solution. In cases where the contribution's problem was suspect, submitters and core members often engaged in extended discussions about the appropriateness of the code change. For example, the submitter may have attempted to implement a feature that is outside the scope of what the software project should be able to do. This discussion over whether the problem to solve was appropriate was actually a negotiation over the requirements of the software project.

Open source software projects tend to not have formal requirements documents that are created through a formal elicitation process [20]. Instead, requirements in open source projects tend to emerge in forms such as mailing list messages or forum posts as a byproduct of the community discussing the direction and assignment of future code contributions [18]. In our findings we saw a similar method for evolving the requirements of the software project when submitters and core members discussed whether a particular code contribution was appropriate for the software project. In other words, whether the problem that the submitter was trying to solve was a problem within the scope of the project's projected feature set.

Besides submitters and core members, other stakeholders such as the third party developers in the audience were also able to participate in evolving the requirements of the software project by participating in the discussion. This is somewhat similar to how community members in traditional open source projects will communicate their needs through bug reports or feature requests [16]. In this open environment however, we saw that perhaps a wider variety of stakeholders were able to influence the requirements of the software project through discussion.

Future research should examine this connection between software requirements and contribution discussions in more detail. Future tool design may explicitly recognize when requirements are being evolved during discussions and may archive these discussions in a more visible way for the benefit of core members.

6. CONCLUSION

In this work we examined how open source developers discuss and evaluate contributions. We found that when developers raised issues with either the problem the submitter was attempting to solve or the solution that was implemented in the pull request, it provided an occasion to discuss alternative solutions or negotiate requirements. Different stakeholders also attempted to influence the outcomes of contributions through pressuring the core or directly alerting them. The transparent environment in our setting provides specific mechanisms for stakeholders in the audience who are outside of the submitter and project core team to participate in the evaluation process.

We found unexpected outcomes for contributions where though a submitter may have their pull request rejected, the core team still fulfilled the technical goals of the submitter in some other way. We also found that the submitter's level of prior interaction on the project changed how core and audience members interacted with the submitter during discussions around contributions.

Our results inform the design of notification and discussion mechanisms for large-scale collaboration where a wide variety of stakeholders participate in evaluation discussions around code contributions. Our findings may also inform how distributed developers negotiate software requirements during code contribution evaluation discussions. Future work should investigate how different kinds of event notification mechanisms influence participation in contribution discussions. Ideally, all legitimate interests should be able to enter the discussion, with notification mechanisms alerting third party stakeholders of relevant discussions. Since submitters also rally support as an effective tactic, more systematic ways of showing support for a change, and perhaps helping to prioritize it relative to other possible changes might also prove useful. Finding ways to identify when conflict resolution mechanisms might also facilitate better and less disruptive ways to handle difficult decisions. Finally, since social relationships seem to have an impact, various mechanisms for visualizing these connections or making them more salient might also impact these negotiations.

7. ACKNOWLEDGMENTS

This material is supported by the Center for the Future of Work at Carnegie Mellon University's Heinz College and by the National Science Foundation under awards IIS1111750 and ACI 1322278.

8. REFERENCES

[1] 10 Million Repositories - GitHub: *https://github.com/blog/1724-10-million-repositories*.

[2] Bryant, S.L., Forte, A. and Bruckman, A. 2005. Becoming Wikipedian: transformation of participation in a collaborative online encyclopedia. *Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work* (New York, NY, USA, 2005), 1–10.

[3] Burke, M. and Kraut, R. 2008. Mind Your Ps and Qs: The Impact of Politeness and Rudeness in Online Communities. *Proceedings of the 2008 ACM Conference on Computer Supported Cooperative Work* (New York, NY, USA, 2008), 281–284.

[4] Corbin, J. and Strauss, A. 2008. *Basics of qualitative research: Techniques and procedures for developing grounded theory*. Sage.

[5] Crowston, K., Wei, K., Howison, J. and Wiggins, A. 2008. Free/Libre open-source software development: What we know and what we do not know. *ACM Comput. Surv.* 44, 2 (Mar. 2008), 7:1–7:35.

[6] Dabbish, L., Stuart, C., Tsay, J. and Herbsleb, J. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work* (New York, NY, USA, 2012), 1277–1286.

[7] Ducheneaut, N. 2005. Socialization in an Open Source Software Community: A Socio-Technical Analysis. *Computer Supported Cooperative Work (CSCW)*. 14, 4 (2005), 323–368.

[8] GitHub: *http://github.com*. Accessed: 2013-08-21.

[9] Gousios, G., Pinzger, M. and Deursen, A. van 2014. An Exploratory Study of the Pull-based Software Development Model. *Proceedings of the 36th International Conference on Software Engineering* (New York, NY, USA, 2014), 345–355.

[10] Halfaker, A., Kittur, A. and Riedl, J. 2011. Don't Bite the Newbies: How Reverts Affect the Quantity and Quality of Wikipedia Work. *Proceedings of the 7th International Symposium on Wikis and Open Collaboration* (New York, NY, USA, 2011), 163–172.

[11] Ko, A.J. and Chilana, P.K. 2011. Design, Discussion, and Dissent in Open Bug Reports. *Proceedings of the 2011 iConference* (New York, NY, USA, 2011), 106–113.

[12] Kraut, R.E. and Resnick, P. 2012. *Building Successful Online Communities: Evidence-Based Social Design*. MIT Press.

[13] Von Krogh, G., Spaeth, S. and Lakhani, K.R. 2003. Community, joining, and specialization in open source software innovation: a case study. *Research Policy*. 32, 7 (Jul. 2003), 1217–1241.

[14] Lerner, J. and Tirole, J. 2002. Some Simple Economics of Open Source. *The Journal of Industrial Economics*. 50, 2 (2002), 197–234.

[15] Marlow, J., Dabbish, L. and Herbsleb, J. 2013. Impression formation in online peer production: activity traces and personal profiles in github. *Proceedings of the 2013 conference on Computer supported cooperative work* (New York, NY, USA, 2013), 117–128.

[16] Mockus, A., Fielding, R.T. and Herbsleb, J.D. 2002. Two case studies of open source software development: Apache and Mozilla. *ACM Trans. Softw. Eng. Methodol.* 11, 3 (Jul. 2002), 309–346.

[17] Rigby, P.C., German, D.M. and Storey, M.-A. 2008. Open source software peer review practices: a case study of the apache server. *Proceedings of the 30th international conference on Software engineering* (2008), 541–550.

[18] Scacchi, W. 2004. Free and open source development practices in the game community. *Software, IEEE*. 21, 1 (2004), 59–66.

[19] Scacchi, W. 2007. Free/Open Source Software Development: Recent Research Results and Methods. *Architectural Issues*. M. V Zelkowitz, ed. Elsevier. 243–295.

[20] Scacchi, W. 2002. Understanding the requirements for developing open source software systems. *Software, IEE Proceedings*- (2002), 24–39.

[21] Strauss, A.L., Corbin, J. and others 1990. *Basics of qualitative research*. Sage Newbury Park, CA.

[22] Teasley, S., Covi, L., Krishnan, M.S. and Olson, J.S. 2000. How Does Radical Collocation Help a Team Succeed? *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work* (New York, NY, USA, 2000), 339–346.

[23] Tsay, J., Dabbish, L. and Herbsleb, J. 2014. Influence of Social and Technical Factors for Evaluating Contribution in GitHub. *Proceedings of the 36th international conference on Software engineering* (2014), In preparation.

[24] Viégas, F.B., Wattenberg, M. and Dave, K. 2004. Studying Cooperation and Conflict Between Authors with History Flow Visualizations. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2004), 575–582.

[25] Van Wendel de Joode, R. 2004. Managing conflicts in open source communities. *Electronic Markets*. 14, 2 (2004), 104–113.