Work Practices and Challenges in Pull-Based Development: The Contributor's Perspective

Georgios Gousios
Radboud University Nijmegen
Nijmegen, the Netherlands
g.gousios@cs.ru.nl

Alberto Bacchelli
Delft University of Technology
Delft, the Netherlands
a.bacchelli@tudelft.nl

ABSTRACT

The pull-based development model, which is gaining widespread popularity with open source systems, offers a novel way to contribute to projects. In this paper, we analyze how contributors experience this model investigating the motivations that drive contributors, their work practices behind the scenes, and the challenges they face. We set up an exploratory qualitative study involving a large-scale survey of more than 650 contributors and analyze it in the light of previous research on contributors' practices and the integrator's role in the pull-based model. Our key findings indicate that motivations are similar to those in other models, but the better traceability for contributions make it more appealing to create a code portfolio, contributors have a controversial relation with awareness, and contribution challenges are mostly social in nature and are exacerbated by the high-volume, sparse, and asynchronous nature of the pull-based model. Moreover, communication within pull requests is limited to low level concerns. On these insights we provide recommendations to practitioners and discuss implications.

1. INTRODUCTION

Open collaborative software projects, such as many open-source software (OSS) systems, leverage external contributors with diverse expertise to add value to a project [48]. In particular, the successful growth and maintenance of OSS depends on creating a sound community of core developers and external contributors [18].

Distributed software development projects employ collaboration models and patterns to streamline the process of integrating incoming contributions [38]. Classic forms of code contributions to collaborative projects include change sets sent to development mailing lists [5] or issue tracking systems [4] and direct access to the version control system of a project [18]. The underlying processes of contribution evaluation have been thoroughly analyzed under different angles [54, 43, 42].

The pull-based development process is a recent form of distributed software development [26]. In this model, contributors do not have access to the main repository, instead, they fork it, make their changes independently, then create a pull request (PR) with their proposed changes to be merged in the main repository. The project's core team (from hereon: *integrators*) is then responsible for evaluating the quality of contributions, proposing corrections, engaging in discussion with the contributors, and eventually integrating them to the main development line or rejecting them. Pull requests are offered by widely used social coding sites, such as GitHub [21], Bitbucket [7], and Gitorious [22]. The popularity of this model is constantly growing with at least 800 thousand collaborative projects (or half of all collaborative projects) adopting it on GitHub alone.

Previous research on the pull-based model analyzed the lifetime characteristics of pull requests [26], the macroscopic factors that lead to contribution acceptance [26, 49], how contributions are evaluated through discussions [50], and the working habits and challenges faced by the integrators [27]. However, little is known of the motivations, work habits, and challenges of contributors to projects that adopt the pull-based model. Understanding these aspects can provide insights for both practitioners and researchers. Projects' developers and leaders can take action against the barriers that contributors encounter and can improve weak aspects of their process; researchers can focus on contributors' challenges to inform the design of better tools and processes to support contributors' work.

To obtain this understanding, we conduct an exploratory qualitative investigation on what motivates top contributors to submit PRs, how they prepare them, and what challenges they face in this contribution model. Since GitHub hosts diverse projects developed by many different programmers, it gives the opportunity to learn from many diverse cases. We do so using surveys, whose questions are based on results from previous literature and our previous experience on this field. We conduct a two-round (pilot and main) survey with 23 and 760 respondents respectively.

Through our investigation, we report on a number of findings including: Motivations for contributing with the pull-based model are in line with that of other models; but the former has the advantage of offering effective traceability between contributions and authors, thus making it a more appealing choice for building code portfolios. Contributors have strong interest in maintaining awareness on what happens in the project to get inspiration and avoid duplicated work, but do not make effort to share their work before submitting PRs. Hardest and most frequent challenges encountered by contributors are social in nature; these are mostly related to low responsiveness of integrators, difficulties in communicating change rationale, and a not empathetic attitude. The pull-based model seems to exacerbate some of these issues, due to its inherent high-volume of requests, information spread in multiple parallel branches, and asynchronous interaction. Moreover, the communication within PRs, although effective for discussing low-level issues shows its limit for other types of contributors' communication needs. Finally, barriers for new contributors, imposed by the model, are mostly due to the learning curve for adopting the necessary tools.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...$15.00.

2. BACKGROUND AND RELATED WORK

OSS projects form online collaborative communities [14] where developers seeking to participate submit contributions, usually as source code patches. A widely accepted view of organizing OSS communities with respect to contributions is the onion model [54]: A core team (perhaps including a leader) receives contributions and determines their fate based on technical merit or trust.

Contributor motivation is a well explored research topic [34, 54, 33, 45, 2]. In one of the earliest studies, Ye and Kishida [54] proposed learning as a core reason for contributing. Lakhani and Wolf [33] proposed extra motives, such as satisfaction and sense of community duty. Shah [45] proposes that having fun and taking challenges are key motivating factors. Fang and Neufeld [16] find that continuous learning and identity building is what drives sustained, rather than ephemeral, contribution activity in OSS communities. Moreover, developers select projects to contribute over others for reasons that have to do with licensing [44] and technical quality [35]. In a meta-analysis of OSS literature on contributor motivation, Androutsellis-Theotokis *et al.* [2, Chapter 9] consolidated existing work and identified 5 high-level aspects of motivation: intrinsic (relating to hedonistic motives), extrinsic (satisfaction obtained through various forms of external compensation), political (ideological beliefs about the value of OSS), social (relating to the added social value the contributions generate), and technical (need to engage with new technologies). Contribution to OSS projects is not only directed by motivations intrinsic to developers; several projects are actively trying to entice developers to contribute as it is to their direct interest (especially for commercially-led projects) to maintain an active and expanding community.

Despite the best project intentions, newcomers to OSS communities occasionally face challenges. Steinmacher *et al.* [48] analyzed the related work and identified 58 barriers, which are, interestingly, mostly related to how the projects work in terms of social aspects, such as community engagement and need for orientation.

After contributions have been submitted they must also be evaluated. In an early study, Mockus *et al.* [38] described the commit-first contribution evaluation pattern: code must be in the repository before it is reviewed. Rigby and Storey examined the peer review process in OSS mailing lists and found that developers filter emails to reduce evaluation load, prioritize using progressive detail within emails containing patches, and delegate by appending names to the patch email recipients [43]. Baysal *et al.* [4] examined contribution evaluation over the bug tracking database; they find that contributions from casual contributors received preferential treatment and they attribute this difference to the size of the contributions (*i.e.*, new contributors submit smaller contributions).

A number of social factors also affect how developers interact with the project community to have their contributions evaluated. Duchneaut found that developers looking to get their contributions accepted must become known to the core team [15]; then, core team members would use the developer's previous actions as one of the signals for judging contributions. Similarly, Krogh *et al.* [51] found that projects permit developers to contribute through established implicit "joining scripts", which are examined to permit access to the main repository based on developers' past actions.

Due to increasing popularity, GitHub in general and pull-based development in particular have recently become a target for researchers of online collaboration and software development practices. Gousios *et al.* [26] provided the first quantitative investigation of the characteristics of contributions through pull requests: They find that contributions are relatively small (20 lines) and processed very fast (submissions are accepted in less than a day). Moreover, both Gousios *et al.* [26] and Tsay *et al.* [49] investigated the factors that underline the acceptance of contributions in pull-based development: Both find similar processes, but different dominating factors (hotness of project area and social distance, respectively).

Contribution evaluation is as important in pull-based development as it is in traditional OSS practices. Pham *et al.* [40] reported initial qualitative evidence on how integrators assess contributions, by focusing on the evaluation of testing practices rather than the pull-based development mode. In a survey of 750 integrators of busy projects, Gousios *et al.* [27] find that integrators struggle to maintain the quality of their projects and have difficulties with prioritizing contributions to be merged and identifying the factors by which they judge the contributions' quality. Tsay *et al.* [50] focus on how discussions affect contribution evaluation in the pull-based development model; they find that stakeholders external to the project may influence the evaluation discussions, while power plays are in effect. Social signals also play an important role: Marlow *et al.* [36] found that core members form an impression of the quality of incoming contributions by using social signals, such as the developer's coding activity and the developer's social actions (*e.g.*, following other developers).

The pull-based development has been found to help projects to engage better with community and consequently attract more contributions. In a study of 40 integrators, Dabbish *et al.* [10] finds that transparency drives collaboration, as social inferences (around commitment, work quality, *etc.*) allows developers to more effectively deal with incoming contributions. Similarly, integrators of three large OSS projects hosted on GitHub, interviewed by McDonald and Goggins, reported that the introduction of the pull-based development model has allowed them to become more democratic and transparent and to attract a greater participation, resulting in doubling the number of contributors [37].

Overall, all studies involving the pull-based development model have focused on projects and integrators. In this paper, we complement this view by analyzing the other part of the equation in the pull-based model, *i.e.*, contributors.

3. RESEARCH METHOD

In this section we present our research questions and describe our research method.

3.1 Research questions

Our examination of the literature revealed that while several researchers examined how developers collaborate in the context of OSS or, more recently, GitHub, no work has examined yet how contributors perceive the pull-based development model. With this model rapidly rising in popularity, it is important to expand our understanding of how it works in practice and what challenges contributors face when using it. Consequently, our overall research goal is to gain an in-depth understanding of the working practices and challenges of contributors in the pull-based development model.

All studies on contributor motivation took place before the pull-based development model was introduced and view contribution as an act of heavy commitment to a project community. However, the pull-based development model facilitates a more casual relationship with projects: The simplicity of sending a pull request and participating in the subsequent discussion has given raise to phenomena such as drive-by commits [40, 36], where developers submit small fixes without expecting any (or at least, limited) compensation or recognition. This calls for a re-investigation of what motivates developers to contribute using the pull-based model, therefore we define our first research question as follows:

RQ1: What motivates the contributors of projects that use the pull-based development model?

After exploring contributors' motivations, we proceed to examine how developers prepare their contributions. Current work, both generally on OSS and on the particular case of the pull-based development, only examines what happens after the contribution has been submitted to the project. In earlier work [26], we found that, despite the increased transparency that social coding sites afford, many contributions are still rejected as duplicate, conflicting or superseded. Do contributors leverage transparency in the same ways that integrators do [10]? Do they communicate prior to submit a pull request or communication is only post-submission? Moreover, contribution quality is a major concern for integrators [27]: Is there a match between what integrators and contributors use and what factors they examine to assess the quality of contributions? This motivates our second research question:

RQ2: How do contributors prepare (for) a pull request?

To ease analysis we refine our question in the following two:

RQ2.1: What happens before and after coding a PR?

RQ2.2: How do contributors assess code quality of a PR?

Finally, we explore the challenges that contributors encounter in the pull-based model. We also consider the barriers hindering new contributors to join. Together with the analysis of the process, this exploration is important to guide future work on this model:

RQ3: What are the challenges of contributing through the pull-based development model?

3.2 Study Design

Our study follows a mixed-method approach [9], mostly collecting qualitative but also quantitative data. We collected data in two rounds: In the first, we run a pilot survey on a limited number of selected contributors; this allowed us to verify the clarity of our questions and identified emerging themes to further explore (*i.e.*, motivations for contributing and barriers for newcomers). In the second round, we sent the survey—augmented with questions addressing the themes emerged in the first round—to a large pool of contributors, identified by quantitative results for each project.

Since our aim is to learn from a large number of projects, we used surveys, a data collection approach that scales well [17].

Survey Design. Both the pilot and the final survey are split into three logical sections: demographic information, multiple choice or Likert-scale questions, and open-ended questions. The open-ended questions are intermixed with multiple choice ones; usually, the contributor had to answer an open-ended question and then a related one with fixed answers. To further elicit the contributor's opinions, in all questions that had predefined answers but no related open-ended question, we included an optional 'other' response. Finally, throughout the survey, we intentionally used even Likert scales to force participants to make a choice. Excluding demographic questions, the survey included overall 4 open-ended questions, 4 Likert scale questions with an optional open-ended response and 11 multiple choice questions (5 with an optional field). The respondents could fill in the survey in about 10 minutes.[1]

[1]Survey questions available at: Georgios ►Add TR link◄

Attracting participants. Previous work [26, 31] showed that most GitHub repositories are inactive, single-user projects. To ensure that our sample consists of repositories that make effective and large scale use of pull requests, we selected all repositories in the GHTorrent dataset [25] that have received at least one pull request per week through the year 2013 (3,400 repositories). For each repository, we extracted the top 3 pull request contributors, by the number of pull requests that they have contributed and we sent them an email, if their address was registered with GitHub.

For the pilot phase, we emailed 445 of those contributors randomly and received 32 answers (7% answer rate). For the data collection phase, we emailed 4,172 contributors from the remaining projects and received 760 answers (18% answer rate). The survey was published online and its web address was sent by personal email to all participants. To encourage participation, we created a customized project report for each of the emailed contributors. The report includes plots on the project's performance in handling PRs (*e.g.*, mean close time) on a monthly basis. The reports for all projects have been published online [24] and widely circulated among developers. We did not restrict access to the survey to invited users only; several survey respondents forwarded the survey to colleagues or advertised it on social media (Twitter) without our consent. After comparing the response set with the original set of projects, we found that 25% of the responses came through third party advertising. The survey ran from April 14 to May 1, 2014.

Respondents. The majority of our respondents self-identified as project contributors (76%), 65% working for industry. Most (68%) have more than 7 years of software development experience and considerable experience ($>$ 3 years) in geographically distributed software development (59%).

3.3 Analysis

We applied manual coding on the 4 open-ended questions as follows: initially, the two authors individually coded a different set of 50 (out of 760) answers for each question. At least one and up to three codes were applied to each answer. The extracted codes were then grouped together and processed to remove duplicates and, in cases, to generalize or specialize them. The new codes were then applied on all answers. When new codes emerged, they were integrated in the code set. On average, 20% more codes were discovered because we decided to code the full dataset.

In the survey, we asked the contributors to optionally report a single repository to which they contribute many pull requests and their GitHub user name. 95% (722) and 81% (610), respectively, of the respondents did so. Many of the responses (126) did not directly match to a GitHub repository for reasons that ranged from spelling mistakes to using wild card names (*e.g.*, jenkinsci/*) to denote contributions to multiple repositories. We corrected the repository names as follows: we first used GitHub's search functionality to locate repositories whose name is similar to the provided one and chose the one that had received pull requests from the user. In case of wild card repository names, we searched the GHTorrent database for all repositories the contributor has submitted pull requests to and selected the one where the contributor has submitted the most pull requests to. We excluded from further analysis answers for which we could not obtain a valid repository name (5 answers) and those that did not include a repository name (38 answers).

After we resolved the repository names, we augmented the survey dataset with information from the GHTorrent database [25]. Specifically, for each project, we calculated the mean number of pull requests per month and the mean number of contributors for the period July 2013 to July 2014. For contributors, we also calculated whether they belong in the top 10 contributor list for the

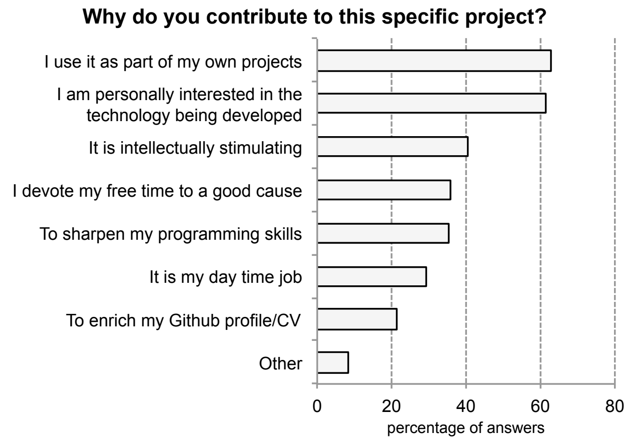

Figure 1: Motivations driving pull request contributors.

specific repository. Using those metrics, and for each one of them, we split the project population in three equally sized groups (small, medium and large).

Finally, to ensure that we our answer set included users that contribute primarily or exclusively through pull requests to the indicated repository, we used one of the fixed answer questions (*Q9: How do you contribute code to the project?*) as a further demarcation point. Consequently, we filtered out 77 respondents who did not indicate contributions exclusively through pull requests or through branch-to-branch pull requests. The final answer set, which we analyze below, contained 645 answers.

4. RESULTS

This section presents the results of our exploratory investigation. When quoting survey respondents, we refer to the individual contributor using a [RX] notation, where X is the contributor's ID.

RQ1: What motivates contributors?

Our first research question seeks to surface what motivations drive contributions to projects that use the pull-based model. We map it to a closed-question in our survey and we offer 7 non-mutually-exclusive closed answers (based on our analysis of the related literature) and an open text field to specify 'other' reasons. Figure 1 summarizes the results.

Only 10% of the respondents used the 'other' field, thus indicating that most motivations for contributing in the pull-based model are in line with those found for different contributions model. By manually coding the open answers, we found that respondents mostly better explained their closed choices. In the following we describe the main themes emerged from the answers to this question (*i.e.*, usage, natural interest and personal growth, and volunteering and paid job) adding quotes from the answers to the 'other' field.

Usage.

The main motivation for contributing to a project is its usage. More than 60% of the respondent chose this answer and 29% of those who added 'other' motivations reiterated on this concept. This usage can be a dependency from another project the contributor is developing (*e.g.*, "*Due to using [application] for my job*" [R288] and "*I have contributed code to integrate GitHub to our commercial software*" [R606]), but it can also correspond to consuming the project as an end-user (*e.g.*, "*[application] is very nice program, that I use daily*" [R571], "*It's a Firefox addon and I personally use it*" [R634]).

When the reason for contributing is usage, it is often connected to contributions in form of bug fixing: "*fixing bugs that personally affect me*" [R35], "*it had some errors that prohibited* [our company] *progress*" [R64] and "*I use the project as part of my work, so if I find and fix a bug*" [R189].

Natural interest and personal development.

Another relevant set of motivations for contributing to a repository regards natural interest and personal development. Almost 60% of respondents are interested in the technology being developed in the projects they contribute to; in the words of one respondent: "*[technology] in general is for the most part very interesting to develop*" [R608]. In particular, 35% of the respondents confirms that contributing to a project help them sharpen their programming skills, *e.g.*, "*My full time job is a non-coding manager. Occasionally contributing bits of code to my teams'* OSS *projects keeps me in touch and shakes off the rust*" [R466], "*I use it to learn from by reading others' code*" [R620]. Approximately 40% contribute because they find it intellectual stimulating. Approximately 35% of the respondents related contributions to personal career development: 23% of the respondents selected the closed answer about enriching their profile/CV (*e.g.*, "*Making contributions to [project] makes it easier for me to get new clients*" [R121]); some contribute to projects as part of their involvement in education and research (*e.g.*, "*This is my research project*" [R596], "*I also use it to teach*" [R244], and "*[it is] part of my master's thesis*" [R578]); and some answers indicate that contributions to GitHub indeed benefit career growth ("*My contribution to [projects] allowed me to obtain a job within my favorite subjects*" [R437], "*I believe in giving back to a community responsible for me having a great career*" [R424]).

Volunteering and paid job.

Contributions are also motivated by either economical interest or the choice of taking part for free in a task for a good cause. About 30% of the respondents make contributions because it is their daily job or part of it (*e.g.*, "*While it isn't my full day job, I do allocate a small part of my work to contribute to [project] (as we use it at work as well)*" [R317], "*Part time student job*" [R594]), a slightly higher percentage (ca. 33%) are motivated to volunteer free time to a good cause (*e.g.*, "*Like to help others*" [R708], "*The open source spirit, if you see a bug or area of improvement, contribute to the main source code so other can enjoy the collaborated rewards*" [R186], and "*I enjoy coding for it and the community*" [R221]). Seven respondents specify in the free text that they contribute to a project because they are the funder, owner, or main maintainer, though they do not specify whether it is volunteer or paid work.

RQ2.1: What happens before/after coding?

With this research question we seek to understand which practices contributors put in place after they decide to create a pull request and before they submit it, particularly before and after the actual coding. We get this understanding with different survey questions.

Before coding.

To learn what happens *before* coding, we provide the respondents with a set of 7 questions (based on our analysis of literature and our experience) with 4-level Likert-scale; Figure 2 reports on the results. Only 24 (3%) respondents added information using the 'other' field, mostly providing clarifications. Results show that, in general, contributors tend to conduct all the mentioned activities; as one developer put it: "*These are all reasonable things to do*"

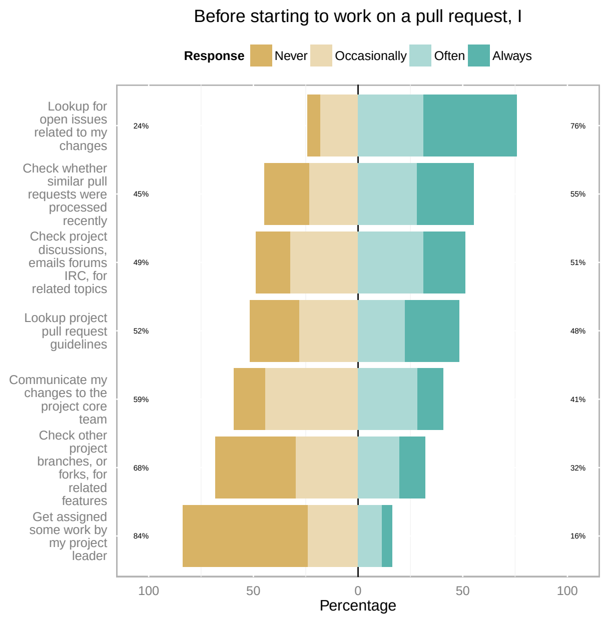

Figure 2: Practices before coding a pull request.

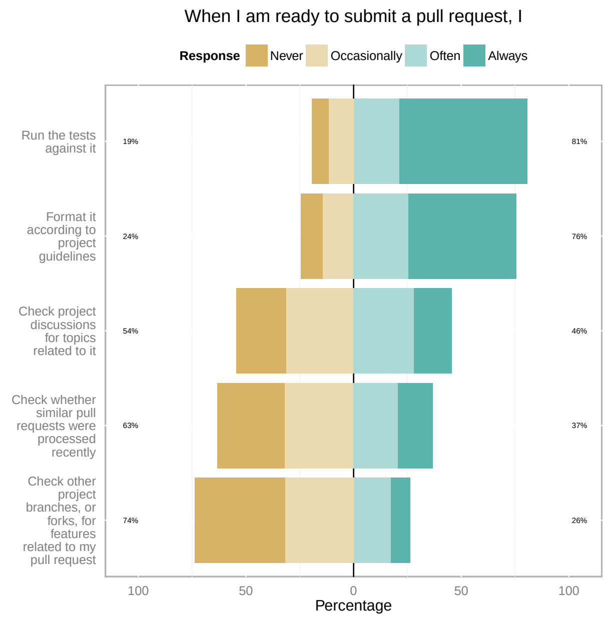

Figure 3: Practices after coding a pull request.

[R490]. Nevertheless, the activities receive different emphasis: In particular, contributors report to conduct practices most related to increasing their *awareness* [13]: They check whether similar previous work had already been done, by consulting (in this order) the issue tracking system, previous pull requests, project communication channels, and external branches/forks. In the 'other' field, respondents explain that the sources are checked both to get inspiration from similar work and to ensure that the work is not going to be duplicated effort (*e.g.*, "*I always create a Bug in Bugzilla to track the work if there is no existing bug*" [R51]). Top experienced contributors explained that they do not need to update awareness, because "*I almost always know what's going on [in the issue tracker], on the mailing lists, in the PR queue, so I have an idea how relevant my PRs are long before I start working on them.*" [R439]

Results in Figure 2 highlight a noteworthy disconnection. Although contributors deem important and spend time checking for work related to the pull request they want to code, once they start coding most of them report to communicate never or occasionally the intended changes with the core team. In other words, information consumption is not supported by information generation: They find important to be aware of what is going on in the project, but they do not personally invest time to increase the overall awareness.

After coding.

Figure 3 summarizes the activities contributors report to conduct just before submitting a pull request, when the code is fully developed. The 'other' field was filled by only a few participants (1%), mostly to clarify their previous choices.

When the coding is finished and contributors are ready to submit a pull request, most developers declare to *not* recheck whether similar work has been accomplished in the meanwhile; in contrast to what they report to do before starting to code. The activities that developers describe as the most frequent before submitting a pull request are formatting the code according to the project's guidelines and running tests against the completed code. Some respondents complain about improper support to these activities by the project (*e.g.*, "*I WOULD run tests and format according to guidelines but there are no tests or guidelines on this project*" [R5], "*no tests available in this repo, but normally I would run tests*" [R1]).

We specifically inquiry about the communication means contributors use, when they decide to communicate on the change. The summarized results are presented in Figure 4. Many respondents explain that they start opening an issue in the tracker or a new pull request, or both. Less frequently they use emails or more synchronous communication channels (*e.g.*, IRC or instant messaging). The respondents adding information in the 'other' field (6%), mainly specify the communication channel they use: Many mention forums (*e.g.*, "*online forums for the project*" [R499]), others IRC-like solutions (*e.g.*, Gitter [R76,77,399]) or project managements tools (*e.g.*, "*Project Management tool such as VersionOne*" [R61]), a few report to use email based communication (*e.g.*, "*Mailing list to get the opinion of the community and core team*" [R286]).

RQ2.2: How is contributions' quality assessed?

Examining the quality of their on contributions is very important for contributors: Contribution quality is the number one factor integrators examine to decide whether a contribution will be accepted or not [27]. To assess how contributors examine their contribution quality, we asked them a compulsory open ended question. While the question was specific on *how* contributors evaluate the quality, the analysis of the results also revealed *what* contributors examine in their PRs. Figure 5 summarizes the results of coding the answers.

What.

One of the top priorities for contributors when examining pull request quality is compliance, which had many manifestations in our response set. The most common one was compliance to project PR or coding guidelines (*e.g.*, "*By following the contribution guidelines for a PR of that repository.*" [R164]); contributors also try to comply to *de facto* guidelines manifested in the original repository,

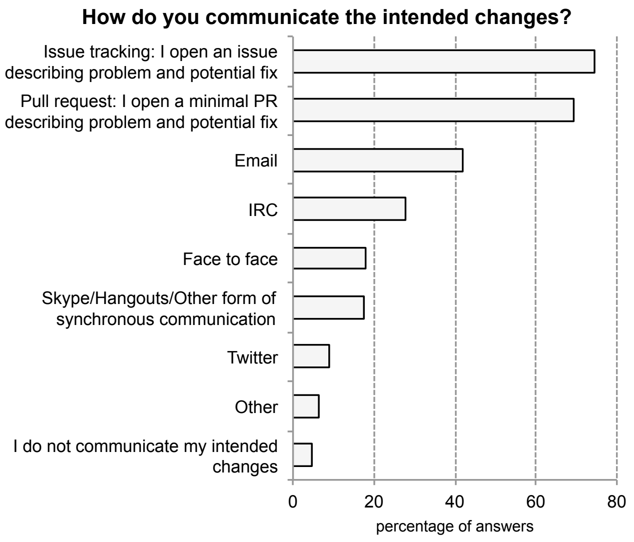

Figure 4: How contributors communicate about the changes.

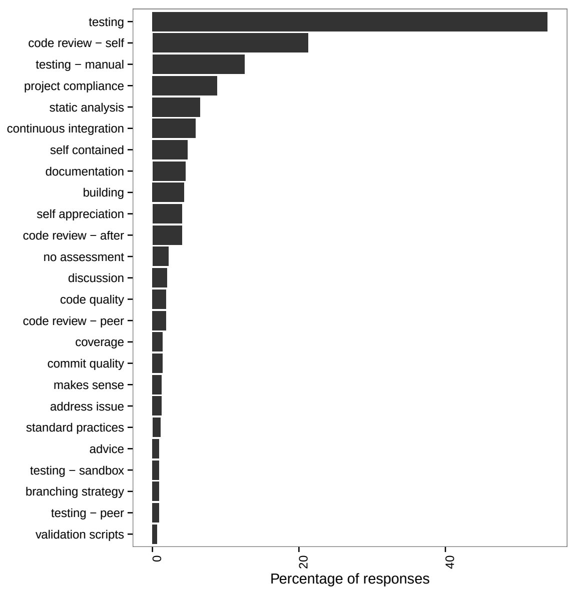

Figure 5: How contributors determine their PR quality.

mainly relating to code formatting (*e.g.*, [R105,584]) and design (*e.g.*, [R252]); another compliance form is adherence to *standard practices*: Contributors try to increase their chances of acceptance by following language code styles and design principles.

On a related note, contributors examine two technical quality aspects: *code quality* and *commit quality*. Code quality is usually assessed subjectively, by examining factors such as readability, clarity, and whether it is minimal ("*the code* [...] *contains only the minimal amount of code change necessary to implement the feature*" [R15]). Several contributors reported to strive to make high quality commits: For some developers this means that commits are "*atomic, and* [can be] *merge*[d] *at time of sending*" [R74], while others assume a more aesthetic view: "*Are the commit messages clear and do the commits in the PR tell a story?*" [R74].

In pull-based development, PRs are usually submitted to projects without prior planning. To increase the chances of acceptance, contributors eagerly examine the pull request's *suitability*, by analyzing whether the PR fully *addresses the issue* it is trying to solve. The term 'issue' is used in the broader sense of an existing problem the developers are addressing, *e.g.*, [R27,538], even though in some cases it is associated with existing bug tracker issues [R306].

Contributors strive for their pull requests to be *self contained* (*e.g.*, "*It should be focused of the feature to implement (or bug to fix). Nothing unrelated to the topic should be in there.*" [R52]) and also try to ensure that the *documentation* of both their code and the pull request eases the comprehension of the PR and meets assumed project standards (thereby enhancing compliance).

How.

By far the most common mechanism that contributors use to assess the quality of their contributions is *testing*. More than 60% of the responses mentioned a form of automated testing (running on the developer's workstation or *continuous integration* servers) in the employed mechanisms. 10% of the contributors using automated testing report to also perform *manual testing* of their changes, by running the contributed code in their projects (*e.g.*, "*Testing it works correctly, on production if needed.*" [R95]), in specialized *sandbox* environments or by their *peers* (*i.e.*, collocated colleagues [R125] or other project developers [R198]). Manual testing is almost always complemented by some other testing mechanism: Only 3% of the respondents indicate that they exclusively test their code manually. The practicality of testing for evaluating contributions and complements the integrator's opinion of testing as a contribution evaluation mechanism [27].

On the tool front, contributors use *static analysis* tools for automatically evaluating their contributions (7%). A wide range of static analysis tools have been reported, but what most developers use falls under three categories: linting tools that detect inconsistencies in coding style and target specific programming languages (*e.g.*, PMD in Java), style checkers that highlight formatting inconsistencies with respect to a pre-specified style (*e.g.*, CHECKSTYLE) and formal method based tools to detect logical inconsistencies (*e.g.*, CPPCHECK). As an extra step, contributors *build* the software to ensure that the compiler will catch simple errors. Building usually invokes the project's test suite.

A complementary examination mechanism contributors use is *code review*, usually (20%) performed by the contributors themselves, before they submit a PR. During code review, the developers examine properties in the source code and documentation as discussed above. When conditions allow it, contributors will ask their *peers* to do a code review before they submit a pull request; the peers can be colleagues (*e.g.*, [R57,172]) or other members of the project community [R382]. Some contributors (4%) rely on code reviews by the project owners or the community after they submit the pull request ("*Others, which have commit access to the repository check the changes if there are any braking changes.*" [R399]). A related response (4% of the respondents) is that they do not explicitly assess the quality of their pull request, because they either believe this is the project owner's responsibility (*e.g.*, [R217,392]) or they bet on immediate acceptance (*e.g.*, [R245]).

Finally, 3.5% of the contributors report that they examine the quality of pull requests using their own *experience* (*e.g.*, "*I try to look at what I'm presenting as if someone else had written it and ask myself if I'd hate dealing with the merge or, having the code in my project with respect to functionality, clarity, and conciseness/elegance (in that order)*" [R238]).

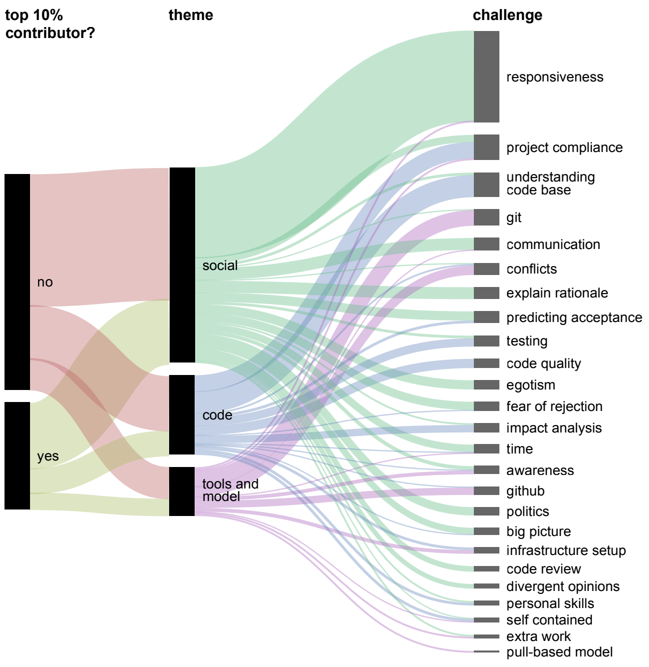

Figure 6: Challenges in contributing pull requests.

RQ3: What are the challenges of contributing?

To find the pain-points emerging when contributing using the pull-based model, we explicitly introduce a mandatory open question in the survey, in which we ask respondents to state the biggest challenge they experience when contributing PRs. By coding the answers, we found that challenges revolved around three main themes: Challenges about writing the *code* for the contribution, challenges on the *tools and model* to be used for submitting the contribution, and challenges pertaining to *social* aspects.

These themes are linked to the finer-grained challenges expressed in the answers. For example, a challenge that emerged is *project compliance*, which in some cases regard the *code* theme (*e.g.*, "*using the project code style*" [R66]) and in others the *social* theme (*e.g.*, "*Not knowing all the rules/process*" [R62]). The results are summarized in Figure 6: From left to right, we first classify the answers on the contributor's rank (*i.e.*, whether it is in top 10% pull request contributor or not), then we show the three main themes and how the answers flow into the specific challenges. The thickness of a line represents the number of responses in the specific category.

Despite we expected the top contributors to be less affected by *tools and model* and *code* related challenges, given their greater experience, to our surprise both types of contributors have very similar distribution of challenges among the three main themes. By far, the theme that is reported by the majority of the respondents to be the most challenging when contributing in a pull-based development model is the *social* one, followed by *code* and *tools and model*, respectively. We analyze the results in this order.

Social aspects.

The *social* theme is connected to most of the reported challenges in different ways. The most prominent one is *responsiveness*: More than 15% of the survey participants find that getting a timely feedback, if any, for their pull requests is hard and they mostly mention people-related causes (*e.g.*, "*The owner of the repo doesn't ever respond to the pull request, and leaves it hanging open forever*" [R15], "[there are] *projects with lots of open pull requests and few actually being accepted*" [R98]). This situation seems to generate frustration in the contributors, up to a point that some mention they might lose interest in the project: "*Malaise and abandonment. Few things are more frustrating than opening a pull request and having it go nowhere*" [R698], "*When contributing to less active projects, it can be really frustrating to have a PR sit untouched for months, since by the time the author gets back to it, I may have given up on it and no longer care*" [R665]. Respondents specify that they would rather receive a clear reject than having no response for their PRs ("*it's annoying to go to the effort of making one and have it ignored... Rejected is better.*" [R85]). Getting feedback on the quality of their work is deemed important for better *predicting acceptance* of future PRs. Particularly, the uncertainty on whether a PR will be accepted is another challenge that can pose difficulties and stress on contributors (*e.g.*, "*When my own code depends on a PR, and I don't know if the PR will get accepted that causes uncertainty and stress.*" [R618])

Poor, delayed, but also general *communication* is reported as another issue. Some contributors specify they find it challenging to *explain rationale* of the their changes and this can affect whether their PRs are thoroughly investigated ("*Sometimes it's hard to explain the need for some changes. Some teams will immediately reject them without analyzing them properly.*" [R491])

A few contributors (*e.g.*, [R190,563]) report *fear of rejection*, since they find it personally embarrassing when their work is judged insufficient by people they have no relationship with. This fear can be exacerbated by the various challenges in interacting with core team members that many respondents report (*e.g.*, "*Fear of looking stupid. Fear of rude response.*" [R190], "*Discouraging project owners*" [R228]). In particular, respondents describe social challenges related to *politics* or how the project is governed (*e.g.*, "*Project owners who really don't want contributions.*" [R122], "*Politics, or project owners not wanting a fix or change, or not actively maintaining it.*" [R526]), *egotism* and general arrogance (*e.g.*, "*People tend to merge only pull requests for issues THEY see as bug.*" [R360], "*Unconstructive/hostile maintainer attitude*" [R536]), and handling *divergent opinions* (*e.g.*, "*getting all* [...] *to agree with a feature you propose in a pull request.*" [R251]).

Furthermore, contributors reported as challenging to find enough *time* to work on the project as they wish (*e.g.*, "*Time to work on complicated issues despite working full time*" [R461]) and to propose contributions that fit in the project's *big picture* and make it grow instead of addressing their needs only (*e.g.*, "*Making sure it's in the interest of the project and not just mine.*" [R183]).

Code aspects.

The *code* theme also permeates a number of challenges. The most frequently reported is *understanding the code base*, including layout and architecture, of the project (*e.g.*, "*Read others code and get understanding of the project design*" [R564]). This problem seems to be often magnified by the project size and lack of documentation (*e.g.*, "[there is] *no guideline or documentation*" [R223], "*Missing knowledge about inner workings of a project* [...] *sometimes caused by missing documentation*" [R561]).

Contributors find also difficult to assess their changes' impact on the rest of the code base (*impact analysis*). Sometimes this is related to their limited understanding of the project (*e.g.*, "*Ensuring that my pull request doesn't have unintended side effects due to not being intimately familiar with the entire code base.*" [R202]) and also to the *social* theme, since *awareness* is not maintained by all contributors (*e.g.*, "*Because of the great complexity of our code,* *contributions by others that are not directly related to my work can nonetheless affect it, and our contributions are not necessarily synced or communicated.*" [R229]). To tackle this challenge and avoid regression, contributors explain they would rely on *testing*, but a proper test suite is not always available and running and developing tests is also a challenge (*e.g.*, "*If there isn't a good testing infrastructure in place, then I'm not sure how to contribute tests related to a pull requests*" [R656]).

Writing PRs with proper *code quality* is mentioned as the only issue by 13 developers. More (7%) report that being compliant with the project style and guidelines is challenging. The *project compliance* on code regards style both at a low typographical level and at higher design level; this challenge also regards difficulties in knowing the format for PRs, commit messages, *etc.* [R277]. Some respondents explain that this challenge is due to tribal knowledge, *i.e.*, information only known within the project team and not explicit to the outside of it (*e.g.*, [R500,659]).

Tools and model.

Respondents report challenges regarding the *tools and model* less frequently. Among those, the usage of *git* and handling *conflicts* between branches are the most prominent ones, especially for seemingly less experienced developers (*e.g.*, "*Usage of git is not intuitive. Especially for me as* [one] *who does not contribute regularly, it is every time a challenge to* [use it]" [R158], "*when projects try to enforce workflow through branches, that is often confusing.*" [R120]). Some respondents mention having problems with the local *infrastructure setup* to be used for development and testing. The few answers explicitly about the *pull-based development model*, mostly regard its learning curve (*e.g.*, [R572]). More respondents mention *github* as a challenge, especially when it comes to have discussions within PRs, thus connecting it to the *social* theme (*e.g.*, "*The comments on a PR can get unwieldy quickly. Without threading it can be hard to follow a conversation*" [R102], "*effectively communicating with other users over github*" [R329]).

Reducing barriers for new contributors.

We complete the answer to our research question by inquiring about what respondents think projects should do to reduce barriers for new contributors. This is mapped in the survey to an open question, which we manually coded. The top 5 barriers that emerged account for more than 50% of the answers. The first barrier (specified by more than 20% of the contributors) regards having good *guidelines* on getting started with the development and testing environment, on code style/formatting conventions, on the contribution process, and on communicating with project owners. The second, third, and fourth barriers follow with a very similar frequency (ca. 15%). In the second barrier, respondents explain that project members should have more *empathy* towards new contributors, providing encouragement, mentoring, fairness, and having an overall positive attitude (*e.g.*, "*Engage in positive, responsive discussion* [...] *Giving a positive first experience goes a long way.*" [R202], "*Maintain a "positive" culture; be friendly, polite etc*" [R73]). The third barrier reiterates on the concept of *responsiveness*: Respondents state that improving it would remove a serious barrier for new contributors, especially as they need more feedback on their work (*e.g.*, "*respond to issues/pull requests/list posts in a timely fashion. Even just acknowledging the issue and suggesting an attack plan is immensely helpful.*" [R504]). The fourth barrier is about the need for a clear project *roadmap* and a comprehensive *task list* with open issues, which should include recommendations for newcomers (*e.g.*, "*They should mark the open issues with the level of difficulty, like these issues are easy and beginners can resolve them*" [R27]). Finally, having better code *documentation* is also reported (ca. 12% of responses) as important to attract new contributors.

5. DISCUSSION

We compare our findings with existing work and discuss recommendations for practitioners and implications for researchers.

Building up a technical CV. Lerner and Tirole were among the first to formalize that contributing to OSS projects is also driven by a *career concern* incentive, which is stronger the more visible the performance to the relevant audience [34]. The pull-based development model makes contributions and their authors easier to recognize than in most other models. GitHub exploits this feature to create public user profiles by including contributions to projects as a sort of portfolio; as Dabbish *et al.* put it: "[it makes] the work visible" [11] On this, various developers discuss whether [1, 52] or not [8] GitHub profiles are and should be the new *de facto* CV for developers. We have strong signals (more than 20% of the respondents report to contribute for career reasons) on the importance of contributions through the pull-based development model on one's career, mostly positive. At the same time, contributors report fear of rejection as one of the challenges of contributing PRs, thus raising the interesting concern on whether too much transparency also hinders contributors' full potential.

Communication. We found that when contributors want to communicate about a change, the preferred mean is no longer the mailing list, which has been considered the hub of project communication in OSS systems for long time [41], but communication means that are more structured and closer to the code artifacts (*i.e.*, a minimal pull request and the issue tracker). Also the IRC-like solutions projects use, such as Gitter, have more advanced features to allow participants to easily link project artifacts (*e.g.*, PRs and issues) from within the discussion using tags. This is in line with the findings by Guzzi *et al.* [28], who observed a shift in OSS developers' communication habits from traditional channels, such as the mailing list, toward more structured channels such as the issue trackers.

On the pull-based model and communication within PRs, contributors' communication needs seem to be only partially met when they use the communication capabilities within the PR's interface (as offered by GitHub, similar to other platforms). Contributors report satisfaction as this form of communication allows to have a clear, structured, and direct communication on the code changes. However, communication within PRs is lacking when developers need to discuss high level concerns ("*In-band communication within the pull request is insufficient for discussing major design changes that might be needed in a new feature*" [R112]), when the number of message within a single PR grows, and when the pull-based development model is strictly enforced, to a point that participants have to channel all communication through PRs only ("*I find it difficult to discuss in advance the changes, since the project is extremely pull-request oriented.*" [R637]). Finally, multiple discussions spread through PRs are problematic for awareness.

Contributors and integrators: shared challenges. By comparing our findings with previous work on the integrator's perspective [27], some challenges emerge as shared, *i.e.*, the two sides of the same coin, thus underlining their overall relevance. Integrators have problems handling and prioritizing the sheer amount of PRs that active projects can attract; contributors complain for the resulting general lack of responsiveness. This is probably due to the pull-based development model that simplifies experimenting and proposing contributions to projects, without reducing the reviewing burden on integrators. Furthermore, contributions' rationale is reported as difficult both by contributors to explain and by integrators to understand, thus raising further concerns on whether the

communication tooling within PRs is appropriate for the task. Finally, integrators and contributors struggle with communicating if a PR is rejected to keep motivation high and show appreciation.
Quality. Contribution quality is a major concern for contributors. It is simultaneously both one of the most frequently reported challenge items and something that developers examine in depth before submitting. Interestingly, quality is also a top priority for integrators as well [27]. A cross examination of the factors that contributors and integrators examine in pull requests reveals that there is again high overlap (compliance/conformance and code quality being top factors). Moreover, automated testing is used in both cases as a commonly accepted way to ensure contribution quality. We hypothesize that this shared understanding of quality is the result of widely accepted technical norms. A positive outcome is that this helps the majority of contributions to be accepted (85%), while rejections are usually not due to technical reasons [26].
Asynchrony. One of the distinguishing characteristics of the pull-based model is asynchrony among the production of a contribution, its evaluation, and its integration. Asynchrony is a cross-cutting concern for both contributors and integrators and its effects are usually detrimental. Asynchrony hinders observability of the current status of the project as a whole and burdens integrators and contributors with extra communication obligations. Indicative is the fact that both contributors and integrators [27] deem responsiveness as a very important challenge, suggesting that more directness is desirable. Interestingly, several high profile companies (*e.g.*, Facebook [23] and Google [32]) moved away from the pull-based model or use strictly bounded code review processes and branching strategies (*e.g.*, Microsoft [6, 3]) to increase development speed for their internal repositories, but still use it for OSS projects.

From a distributed systems theoretical standpoint, mitigating the results of asynchrony is impossible [46]. Integrators and contributors should therefore either agree on minimal communication protocols that increase each other's awareness or, in certain cases, to abandon the pull-based model in favor of more direct loop model.

Awareness. While most contributors report that they examine the issue tracker for existing similar issues or pull requests, previous work report that PRs are often rejected because they are duplicate or superseded [26]. This indicates a critical step between contemplating about a contribution and actually creating it. Moreover, most contributors are not interested to update their knowledge of the project status after coding, before they submit their pull request. This suggests that building and maintaining awareness is of limited interested to contributors even though it is crucial for mitigating the negative effects of asynchrony.

5.1 Recommendations for practitioners

In this section, we present a set of guidelines that integrators and contributors can apply to be more effective when working with the pull-based model. Preliminary guidelines were presented in Gousios et al [26]; here, we improve them by integrating current findings and those of Gousios *et al.* [27].
Integrators. Research has shown [] that it is to the direct benefit of an OSS project to maintain a healthy community. The pull-based model effectively flattens the community structure outside the core team circle and even blurs the line between a core team member and a community member and is therefore very well suited for this purpose. Consequently, integrators should work towards streamlining the contribution process. Initially, the project should include *comprehensive contribution guidelines*; those should at least detail the expected code style, commit format, pull request process, and available communication options. Well-laid out contribution guidelines help contributors format contributions using the expected style and can act as a reference in code review discussion. Moreover, projects should *invest in good tests*: not only contributors gain confidence about their contributions by testing them locally but also contribution evaluation is faster [26]. *Automation* is also important: It should at least cover the *development environment setup* and the *quality evaluation* of contributions. Ideally, in the first case, the contributor should be able to setup a fully working development environment by running a simple command; tools exist (*e.g.*, Vagrant, Docker and Puppet) that allow the programmatic definition of full systems. Integrators should also invest time to setup fully automatic quality evaluation of incoming contributions. In the case of GitHub, external services exist that enable continuous integration (*e.g.*, Travis and Cloudbees) and code quality (*e.g.*, Codeclimate) monitoring on a per contribution basis.

Performance monitoring can help integrators identify hotspots and bottlenecks and improve upon those. Initially, the integrators should compare their project's performance against the norms: Data shows [26] that on average 85% of pull requests are merged and 2/3 of them are merged within a day, so if a project significantly deviates from those, integrators should investigate why. Also, as we found that lingering pull requests comprise the majority of open pull requests [26] and they hinder awareness of project status, it will be beneficial for integrators to monitor slow processed contribution lifelines and promptly close lingering ones. Finally, the pull-based model helps monitoring to be extended to the project's community: How big is it, what portion of it is contributing back and how balanced between core team members and contributors are the discussions on PRs?

Finally, as social challenges are more pronounced than technical ones in all studies, it is important for integrators to address them. Thus, integrators should be both *proactive*, by establishing (and perhaps even documenting) a professional communication etiquette and *reactive* by following discussions and intervene in cases where discussion diverges from the etiquette. Moreover, when possible, integrators should refrain from invoking political discussions and avoid participating in power struggles [50].
Contributors. From an economic standpoint, it is beneficial for the contributor that code contributions are eventually accepted.

For one-off contributions, the contributors should employ strategies that *minimize friction* and *maximize awareness*. Initially, it helps if contributions are *small* and *isolated*. In previous work [53, 4, 26, 27], the size of the change is one of the most important factors related with acceptance: This happens because the impact of the change is more easily evaluated, especially if the change does not cross logical functionality or design boundaries. The contributors should also make their changes *adhere to guidelines* and learn how to use the underlying tools (git), as this saves review time. Awareness can be increased by contacting the development team using real-time communication channels (*e.g.*,IRC) or by following the minimal pull request idiom [2] (depending on project preferences). Finally, contributors should be available after submission to discuss the results of the code review promptly to minimize the negative effects of asynchrony.

For more serious involvement in project communities, *profile building* through a stream of excellent contributions and *participation* in other community activities (*e.g.*, discussion of issues) are essential; integrators both evaluate [36, 27] and prioritize [27] work using a mixture of social signals and developer track records.

5.2 Implications for researchers

Our work uncovered aspects that deserve further study:

[2]https://github.com/blog/1124

Prioritization of PRs. We identified low responsiveness as one of the most recurring challenges experienced by contributors; interestingly, integrators reported problems in prioritizing PRs [27]. Automating the prioritization of PRs would at the same time help integrators better allocating their time and show contributors the status of their PRs with respect to the overall queue. Automated prioritization could take advantage of explicit integrators' preferences (*e.g.*, place bug fixes first), thus making contributors aware of such choices in an automatically generated guideline.

Estimated time for merging. A widespread usability heuristic states that "system should always keep users informed about what is going on" [39]. This is often achieved, for example, through progress bars in application UIs. Contributors' frustration on not knowing the status and fate of their PRs indicates that having a capability for estimating the time for merging a contribution would be valuable. If the estimation engine could provide indication on the most significant factors considered for the prediction, contributors could take advantage of it to understand what could be improved for speeding up acceptance (*e.g.*, splitting a PR into self-contained tasks) and would help keeping contributors decide earlier whether to stick with a project or leave. Previous research estimated merging time for patches [30], closing time for issue reports [20, 55], *etc.* This is a ripe opportunity for the authors of such research to have an additional impact on a wide population of developers.

Untangling code changes. Integrators report that code understanding and reviewing is simplified if code changes pertain to a single, self-contained task [3, 27]; however, integrators have difficulties obtaining such PRs and contributors report creating them to be a challenge. Recently, researchers are proposing automated approaches to split changes into self-contained tasks [29, 12] in the context of mining software repositories. This is an interesting opportunity to apply these methods and integrate them in the pull-based model workflow.

Impact analysis on PRs. All contributors and integrators are interested in knowing the impact of the proposed PRs, beyond the changed code. The pull-based development model is a fine opportunity to provide results of impact analysis research to a broad community and test its effects on the field. Tools' results could be integrated in the pull request interface as an optional service.

Improved communication within PRs. Our work let emerge a number of drawbacks in communication occurring within PRs, despite its advantages of being close to the changed code. Particularly, the communication support should be improved for discussing high-level concerns, for scaling to longer discussions, and for maintaining awareness. Multidisciplinary studies involving user interface designers, communication experts, and software engineering can be designed and carried out to determine how the communication experience within PRs can be improved.

6. LIMITATIONS

We designed our survey with the stated aim of gaining insights on a novel mechanism of collaborating in distributed software development. For closed selection questions, the response categories originated in the literature and our prior experience with working and researching [26, 27] with pull requests and the pull-based model. The ordering and wording of questions were carefully selected to avoid leading the respondent and where validated through (1) consultation with colleagues expert in qualitative research, (2) a formal pilot run, and (3) several mini-runs. Despite our best efforts, this study may be subject to the following limitations:

Internal validity – Credibility. We used coding to classify the contributors' responses in open-ended questions. The coding process is known to lead to increased processing and categorization capacity at the loss of accuracy of the original response.

Moreover, question-order effect [47] (*e.g.*, one question could have provided context for the next one), the open ended questions, as well as social desirability bias [19] (*i.e.*, a respondent's possible tendency to to appear in a positive light, such as by showing they are fair or rational) may have influenced the answers' accuracy.

Generalizability – Transferability. Our selection of projects and, consequently, contributors to survey, while representative of big users of the pull-based model may not be indicative of the *average* project. Previous work [26] found that the median number of pull requests across repositories is 2; in our sample and considering the initial selection of projects, the smallest project had more than 400. We expect that if the study is repeated using random sampling for projects, the results will be slightly different. This may in particular affect the results on obstacles such as low responsiveness and inefficient communication, as average projects do not use pull requests in a high capacity. To reduce other limitations to generalizability of our work, we did not impose other restrictions on the sample projects, such as programming language or use of technologies.

Moreover, GitHub is only one, albeit the biggest, of the code hosting sites featuring the pull-based development model. While this model remains the same across all these sites, the implementation of several GitHub features might influence the developer's opinions of the model. In both our question set and our interpretation of the results, we avoided direct references to GitHub's implementation of the mechanism. However, bias in the contributors' answers could not be completely eradicated, as can be witness by the fact that many open ended answers included direct references to GitHub or tools in its ecosystem (*e.g.*, Travis CI).

7. CONCLUSIONS

Our work studied the pull-based development model from the contributors' perspective. Our goal was to better understand the motivations that drive contributors, the work practices contributors put in place behind the scenes of creating a PR, and the challenges they face with the overall pull-based model. We make the following key contributions:

- A publicly available iteratively-tested survey with questions for eliciting contributors' practices in the pull-based model and the anonymized answers of 645 respondents.
- The set of open-ended questions we manually coded and R analysis scripts for overall data analysis.
- A thorough analysis of the survey answers resulting in answers to our research questions on contributors' motivations, PR's preparation, and open challenges in contributing with the pull-based model.
- A discussion comparing our findings with previous literature and guidelines for practitioners using the pull-based model and data-derived directions for future research.

It is our hope that the insights we have discovered lead to more effectively merging external contributions in practice and improved tools, based on research, to aid developers creating and managing code contributions.

Acknowledgements

We thank Margaret-Anne Storey for her comments and constructive criticism and Anja Guzzi for interesting discussions.

8. REFERENCES

[1] Github is your new resume. `https://news.ycombinator.com/item?id=2763182`. Accessed 2015/03/10.

[2] S. Androutsellis-Theotokis, D. Spinellis, M. Kechagia, and G. Gousios. Open source software: A survey from 10,000 feet. *Foundations and Trends in Technology, Information and Operations Management*, 4(3–4):187–347, 2011.

[3] A. Bacchelli and C. Bird. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 712–721, Piscataway, NJ, USA, 2013. IEEE Press.

[4] O. Baysal, O. Kononenko, R. Holmes, and M. Godfrey. The secret life of patches: A firefox case study. In *Reverse Engineering (WCRE), 2012 19th Working Conference on*, pages 447–455, Oct 2012.

[5] C. Bird, A. Gourley, and P. Devanbu. Detecting patch submission and acceptance in oss projects. In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, MSR '07, pages 26–, Washington, DC, USA, 2007. IEEE Computer Society.

[6] C. Bird and T. Zimmermann. Assessing the value of branches with what-if analysis. In *Proceedings of the 20th International Symposium on Foundations of Software Engineering*, November 2012.

[7] Bitbucket. `http://bitbucket.org/`. Accessed 2015/03/10.

[8] J. Coglan. Why github is not your cv. `https://blog.jcoglan.com/2013/11/15/why-github-is-not-your-cv/`, November 2013. Accessed 2015/03/10.

[9] J. W. Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage Publications, 3rd edition, 2009.

[10] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in Github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, CSCW '12, pages 1277–1286, New York, NY, USA, 2012. ACM.

[11] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Leveraging transparency. *IEEE Software*, 30(1):37–43, 2013.

[12] M. Dias, A. Bacchelli, G. Gousios, D. Cassou, and S. Ducasse. Untangling fine-grained code changes. In *Proceedings of the 22nd International Conference on Software Analysis, Evolution, and Reengineering*, SANER 2015. IEEE Computer Society, 2015.

[13] P. Dourish and V. Bellotti. Awareness and coordination in shared workspaces. In *Proceedings of the ACM conference on Computer-supported cooperative work*, pages 107–114. ACM, 1992.

[14] N. Ducheneaut. Socialization in an open source software community: A socio-technical analysis. *Computer Supported Cooperative Work (CSCW)*, 14(4):323–368, 2005.

[15] N. Duchneaut. Socialization in an open source software community: A socio-technical analysis. *Computer Supported Cooperative Work (CSCW)*, 14(4):323–368, 2005.

[16] Y. Fang and D. Neufeld. Understanding sustained participation in open source software projects. 25(4):9–50, Apr. 2009.

[17] U. Flick. *An introduction to qualitative research*. SAGE Publications, 5th edition, 2014.

[18] K. Fogel. *Producing Open Source Software*. O'Reilly Media, first edition, 2005.

[19] A. Furnham. Response bias, social desirability and dissimulation. *Personality and Individual Differences*, 7(3):385 – 400, 1986.

[20] E. Giger, M. Pinzger, and H. Gall. Predicting the fix time of bugs. In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*, pages 52–56. ACM, 2010.

[21] GitHub. `https://github.com/`. Accessed 2015/03/10.

[22] Gitorious. `http://gitorious.org/`. Accessed 2015/03/10.

[23] D. Goode and S. Agarwal. `https://code.facebook.com/posts/218678814984400/scaling-mercurial-at-facebook/`. Accessed 2015/03/10.

[24] G. Gousios. `http://ghtorrent.org/pullreq-perf/`. Accessed 2015/03/10.

[25] G. Gousios. The GHTorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 233–236, Piscataway, NJ, USA, 2013. IEEE Press.

[26] G. Gousios, M. Pinzger, and A. van Deursen. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 345–355, New York, NY, USA, 2014. ACM.

[27] G. Gousios, A. Zaidman, M.-A. Storey, and A. v. Deursen. Work practices and challenges in pull-based development: The integrator's perspective. In *Proceedings of the 37th International Conference on Software Engineering*, ICSE 2015, 2015.

[28] A. Guzzi, A. Bacchelli, M. Lanza, M. Pinzger, and A. van Deursen. Communication in open source software development mailing lists. In *Proceedings of MSR 2013 (10th IEEE Working Conference on Mining Software Repositories)*, pages 277–286, 2013.

[29] K. Herzig and A. Zeller. The impact of tangled code changes. In *Proceedings of 10th Conference on Mining Software Repositories*, pages 121–130. IEEE, 2013.

[30] Y. Jiang, B. Adams, and D. M. German. Will my patch make it? and how fast?: case study on the linux kernel. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 101–110. IEEE Press, 2013.

[31] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. The promises and perils of mining github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 92–101, 2014.

[32] A. Kumar. `http://www.infoq.com/presentations/Development-at-Google`. Accessed 2015/03/10.

[33] K. Lakhani and R. Wolf. *Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects.* MIT Press, Cambridge, 2005.

[34] J. Lerner and J. Tirole. Some simple economics of open source. *The journal of industrial economics*, 50(2):197–234, 2002.

[35] W. Maalej, H.-J. Happel, and A. Rashid. When users become collaborators: Towards continuous and context-aware user input. In *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, OOPSLA '09, pages 981–990, New York, NY, USA, 2009. ACM.

[36] J. Marlow, L. Dabbish, and J. Herbsleb. Impression formation in online peer production: activity traces and personal profiles in github. In *Proceedings of the 2013 conference on Computer supported cooperative work*, CSCW '13, pages 117–128, New York, NY, USA, 2013. ACM.

[37] N. McDonald and S. Goggins. Performance and participation in open source software on github. In *Extended Abstracts on Human Factors in Computing Systems*, CHI EA '13, pages 139–144, New York, NY, USA, 2013. ACM.

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

[39] J. Nielsen. 10 usability heuristics for user interface design. http://www.nngroup.com/articles/ten-usability-heuristics/, January 1995.

[40] R. Pham, L. Singer, O. Liskin, F. Figueira Filho, and K. Schneider. Creating a shared understanding of testing culture on a social coding site. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 112–121, Piscataway, NJ, USA, 2013. IEEE Press.

[41] E. Raymond. *The Cathedral and the Bazaar - Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly, 1999.

[42] P. C. Rigby and C. Bird. Convergent contemporary software peer review practices. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 202–212, New York, NY, USA, 2013. ACM.

[43] P. C. Rigby and M.-A. Storey. Understanding broadcast based peer review on open source software projects. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 541–550, New York, NY, USA, 2011. ACM.

[44] C. Santos, G. Kuk, F. Kon, and J. Pearson. The attraction of contributors in free and open source software projects. *The Journal of Strategic Information Systems*, 22(1):26 – 45, 2013. Service Management and Engineering in Information Systems Research.

[45] S. K. Shah. Motivation, governance, and the viability of hybrid forms in open source software development. *Management Science*, 52(7):1000–1014, 2006.

[46] J. Sheehy. There is no now. *ACM Queue*, 13(3):1–8, 2015.

[47] L. Sigelaman. Question-order effects on presidential popularity. *Public Opinion Quarterly*, 45(2):199–207, 1981.

[48] I. Steinmacher, T. U. Conte, M. Gerosa, and D. Redmiles. Social barriers faced by newcomers placing their first contribution in open source software projects. In *Proceedings of the 18th ACM conference on Computer supported cooperative work & social computing*, pages 1–13, 2015. To appear.

[49] J. Tsay, L. Dabbish, and J. Herbsleb. Influence of social and technical factors for evaluating contribution in github. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 356–366, New York, NY, USA, 2014. ACM.

[50] J. Tsay, L. Dabbish, and J. Herbsleb. Let's talk about it: Evaluating contributions through discussion in github. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 144–154, New York, NY, USA, 2014. ACM.

[51] G. von Krogh, S. Spaeth, and K. R. Lakhani. Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32(7):1217 – 1241, 2003. Open Source Software Development.

[52] B. Weiss. Github is your resume now. `http://anti-pattern.com/github-is-your-resume-now`, June 2012. Accessed 2015/03/10.

[53] P. Weissgerber, D. Neu, and S. Diehl. Small patches get in! In *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, MSR '08, pages 67–76, New York, NY, USA, 2008. ACM.

[54] Y. Ye and K. Kishida. Toward an understanding of the motivation of open source software developers. In *Proceedings of the 25th International Conference on Software Engineering, 2003*, pages 419–429, May 2003.

[55] H. Zhang, L. Gong, and S. Versteeg. Predicting bug-fixing time: an empirical study of commercial software projects. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 1042–1051. IEEE Press, 2013.