2016 IEEE/ACM 38th IEEE International Conference on Software Engineering

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

Margaret-Anne Storey
University of Victoria
BC, Canada
mstorey@uvic.ca

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

ABSTRACT

The pull-based development model is an emerging way of contributing to distributed software projects that is gaining enormous popularity within the open source software (OSS) world. Previous work has examined this model by focusing on projects and their owners—we complement it by examining the work practices of project contributors and the challenges they face.

We conducted a survey with 645 top contributors to active OSS projects using the pull-based model on GitHub, the prevalent social coding site. We also analyzed traces extracted from corresponding GitHub repositories. Our research shows that: contributors have a strong interest in maintaining awareness of project status to get inspiration and avoid duplicating work, but they do not actively propagate information; communication within pull requests is reportedly limited to low-level concerns and contributors often use communication channels external to pull requests; challenges are mostly social in nature, with most reporting poor responsiveness from integrators; and the increased transparency of this setting is a confirmed motivation to contribute. Based on these findings, we present recommendations for practitioners to streamline the contribution process and discuss potential future research directions.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Version control*; D.2.9 [**Software Engineering**]: Management—*Programming teams*

Keywords

pull-based development, open source contribution, pull request, distributed software development, GitHub

1. INTRODUCTION

Distributed software development projects employ collaboration models and patterns to streamline the process of integrating incoming contributions [36]. The pull-based development model is a recent form of distributed software development [25] that is gaining tremendous traction in the open source software (OSS) world.

ICSE '16, May 14 - 22, 2016, Austin, TX, USA

ISBN 978-1-4503-3900-1/16/05...$15.00

DOI: http://dx.doi.org/10.1145/2884781.2884826

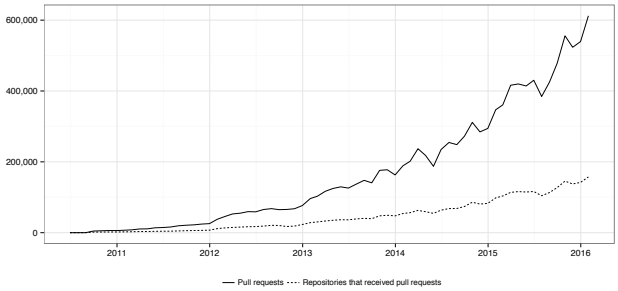

Figure 1: Monthly growth of pull request usage on GitHub.

As Figure 1 shows, its popularity is constantly growing; in January 2016, 135,000 repositories on the GitHub social coding site received more than 600,000 pull requests. In total, 1,000,000 collaborative GitHub projects (*i.e.*, 45% of all collaborative projects) used at least one pull request during their lifetime.

As opposed to more classic ways of contributing (*e.g.*, change sets sent to development mailing lists [4] to issue tracking systems [3] or through direct access to the version control system [17]), in the pull-based model, contributors fork (*i.e.*, locally duplicate) the main repository of the project they want to contribute to, make their changes independently, then create a pull request (PR) to ask that their changes be merged into the main repository. Then the members of the project's core team (the *integrators*) are responsible for evaluating the quality of the contributions, proposing corrections, engaging in discussion with the contributors, and eventually merging or rejecting the changes.

Social coding sites (*e.g.*, GitHub [20], Bitbucket [6], and Gitorious [21]) offer the pull-based development model in conjunction with social media functions, which allow users to subscribe to and/or visualize information about activities of projects and users and offer threaded asynchronous communication within PRs.

To grasp the complexity of the pull-based development model offered by social coding sites, it is necessary to examine it from multiple perspectives. Previous research considered the lifetime characteristics of PRs [25], macroscopic factors that lead to contribution acceptance [25, 45], the barriers faced by first time contributors [44], how contributions are evaluated through discussions [46], and the working habits and challenges faced by *integrators* [27]. Here we present *the contributor's perspective* by investigating contributors' work habits and the challenges they face.

The overall goal with this work is to *understand how contributing to OSS projects works using the pull-based development model* in the context of social coding sites. Understanding the contributor's perspective is needed to reveal weaknesses with the pull-based model and to guide the design of tools and processes to support their work, which is an essential part of the workflow. More-

over, this understanding is required to help project owners address the weak aspects of their development process and to take action against the barriers that contributors face.

To achieve our goal, we performed an exploratory investigation of contributors to projects hosted on GitHub. We conducted an online survey that was answered by 645 top contributors to active projects, and we analyzed the results with traces from GHTorrent [24]. Since GitHub hosts diverse projects developed by many different programmers, it gives us the opportunity to learn from a variety of cases.

We found that contributors have a strong interest in staying *aware* of project status to get inspiration and to avoid duplicating work, but they do not actively try to propagate information about the pull requests they are preparing. The toughest and most frequent challenges encountered by contributors are *social* in nature, mostly related to *poor responsiveness* and a lack of empathy from integrators, and difficulties in communicating change rationale. In particular, the *communication* within pull requests, although effective for discussing low-level issues, appears to be limited for other types of contributors' communication needs. Finally, considering the *transparency* offered by social coding sites using the pull-based model, our respondents reported (in line with previous findings [11]) that building code portfolios is a motivation to contribute. The same transparency seems to encourage developers to review their changes before submission, but also triggers a fear of rejection that might harm their reputation.

2. BACKGROUND AND RELATED WORK

OSS projects form online collaborative communities [15] where developers wishing to participate will submit contributions, usually as source code patches. The onion model is a widely accepted way of organizing OSS communities by contributions [52]: a core team (with an optional leader) receives contributions and determines their fate based on technical merit or trust.

Despite a project's best intentions, newcomers to OSS communities occasionally face challenges. Steinmacher *et al.* [44] analyzed related work and identified 58 barriers, most of which relate to social aspects such as community engagement and the need for orientation.

After contributions have been submitted, they must also be evaluated. In an early study, Mockus *et al.* [36] 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 [41]. Baysal *et al.* [3] examined contribution evaluation over the bug tracking database and found that contributions from casual contributors received preferential treatment, which they attribute 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 in order to have their contributions evaluated. Duchneaut found that developers hoping to get their contributions accepted must first be known to the core team [15]; core team members use a developer's previous actions as one of the signals for judging contributions. Similarly, Krogh *et al.* [49] found that projects permit developers to contribute through established implicit "joining scripts", which may permit access to the main repository based on developers' past actions.

Due to increasing popularity, GitHub and the pull-based development approach have attracted the attention of researchers interested in online collaboration and software development practices. Gousios *et al.* [25] quantitatively investigated the characteristics of contributions in GitHub, finding that contributions are relatively small (20 lines) and processed very quickly (submissions are accepted in less than a day). Moreover, both Gousios *et al.* [25] and Tsay *et al.* [45] investigated the factors that influence the acceptance of contributions in GitHub: both found similar processes but different dominating factors (*i.e.*, '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.* [38] reported initial qualitative evidence on how integrators assess contributions by focusing on the evaluation of testing practices. In a survey of integrators of busy projects in GitHub, Gousios *et al.* [27] found that integrators struggle to maintain the quality of their projects. They experience difficulties prioritizing the contributions to be merged and face challenges identifying factors that will reveal contribution quality. By focusing on how discussions affect contribution evaluation in GitHub, Tsay *et al.* [46] found 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.* [34] 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).

Social coding capabilities, in conjunction with pull-based development, improve how projects engage with community members and attract more contributions. In a study of integrators, Dabbish *et al.* [11] found that transparency drives collaboration as social inferences (around commitment, work quality, *etc.*) allow 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 switch to GitHub allowed them to become more democratic and transparent and to attract more participation, resulting in a doubling of the number of contributors [35].

Overall, all studies involving the pull-based development model have focused on projects and integrators. In this paper, we complement this view by analyzing contributors.

3. RESEARCH METHOD

The overall goal with this research is to *understand how contributing to OSS projects works using the pull-based development model* in the context of social coding sites. Our examination of the literature revealed that our scientific knowledge of OSS and the case of pull-based development in social coding mostly considers project or integrator perspectives, or investigates what happens after a contribution has been submitted to a project.

We know that the pull-based development model facilitates a more casual relationship with projects by making it easier to send a pull request, while social coding features ease participation in any subsequent discussion. This setting has given rise to phenomena such as drive-by commits [38, 34], where developers submit small fixes without expecting any (or at least, limited) compensation or recognition. What is currently lesser known, however, is how contributors prepare (for) a contribution. This motivated our first research question:

> RQ1: How do contributors prepare (for) a contribution in social coding sites using the pull-based development model?

Despite the increased transparency that social coding sites afford, many contributions are still rejected as duplicate, conflicting

or superseded [25]. We were interested in finding out whether contributors leverage transparency in the same ways that integrators do [11]. Do they communicate prior to submitting a PR or does communication occur only post-submission? Moreover, contribution quality is a major concern for integrators [27]: we wished to know if there is a match between what integrators and contributors examine to assess the quality of contributions. We refined our first research question as follows:

RQ1.1: *What do contributors do before and after coding a PR?*

RQ1.2: *How do contributors assess the quality of their PR?*

RQ1.3: *How do contributors communicate about an intended change?*

Subsequently, we were interested in understanding the challenges that contributors experience when working with the pull-based model in GitHub. We also considered the barriers that make it difficult for new contributors to participate. This exploration is needed to guide future work in this area and led to our last research question:

RQ2: What are the challenges of contributing in social coding sites using the pull-based development model?

3.1 Study Design

Our study followed a mixed-method approach [10]. Since our aim was to learn from a large number of projects, we used an online survey as it is a data collection approach that scales well [16]. We enriched the gathered data with traces extracted from GHTorrent [24]. We collected survey data in two rounds. In the first round, we ran a pilot survey with a limited number of selected contributors as it allowed us to clarify our questions and to identify emerging themes we could explore further (*i.e.*, we added a question about motivations for contributing and one question on barriers for newcomers). In the second round, we sent the survey—augmented with questions addressing the themes that emerged in the first round—to several contributors of pull requests on GitHub-hosted OSS projects.

Survey Design. Pilot and final survey were split into two sections: (1) demographic information and (2) open-ended questions intermingled with multiple choice or Likert scale questions. 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. Throughout the survey, we intentionally used even Likert scales to force participants to make a choice. Excluding demographic questions, the final survey consisted of 4 open-ended questions, 4 Likert scale questions with an optional open-ended response, and 11 multiple choice questions (5 with an optional field).[1] The vast majority (95%) of respondents completed the survey in less than 10 minutes.

Sampling projects and candidate respondents. Previous work has revealed that most GitHub repositories are inactive and have a single user [25, 31]. To ensure that our sample consisted of repositories that make effective and large-scale use of PRs, we selected all repositories in the GHTorrent dataset [24] that have received at least one PR per week during the year 2013 (3,400 repositories). For each repository, we extracted the top 3 pull request contributors by the number of PRs they contributed. We sent them an email if their address was registered with GitHub and if they were not integrators in the same repository; we collected 4,617 emails.

[1]The survey questions are publicly available [26].

Attracting participants. For the pilot phase, we randomly selected and emailed 445 of the 4,617 contributors, and received 32 answers (7% response rate). For the main data collection phase, we emailed the remaining 4,172 contributors and received 760 answers (18% response rate, *i.e.*, typical of online surveys in software engineering, where the response rate is usually within the 14–20% range [39]). 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 included 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 [23] and they were 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%), with 65% working for industry. Most (68%) reported more than 7 years of software development experience and considerable experience ($>$ 3 years) in geographically distributed software development (59%).

3.2 Analysis

We applied manual coding [9] on the 4 open-ended questions as follows: initially, the first and last authors individually coded (in a shared online spreadsheet) a different set of 50 (out of 760) answers for each question. At least 1 and up to 3 codes were applied to each answer. Then, the coders met physically, grouped the extracted codes together and processed them to remove duplicates and, in some cases, to generalize or specialize them. The agreed-upon codes were then applied to all the answers (each coder codified 50% of the answers). When new codes emerged, they were integrated in the code set. Another round of code integration followed in a physical meeting, which led to the final result. On average, 20% more codes were discovered in the final integration round.

We asked respondents to optionally include their GitHub user name and report a single repository to which they contribute many PRs; 81% (610) and 95% (722) of the respondents did so, respectively. Many responses (126) did not match to a GitHub repository for reasons that ranged from spelling mistakes to using names with wild cards (*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 was similar to the provided one and then chose the one that had received PRs from the user. When a repository name contained wild cards, we searched the GHTorrent database for all repositories the contributor had submitted PRs to and selected the one where the contributor had submitted the most PRs. We excluded from our further analysis any answers for which we could not obtain a valid repository name (5 answers) and those that did not include a repository name (38 answers).

3.3 Adding Project Metrics

After we resolved the repository names, we augmented the survey dataset with information from the GHTorrent MySQL database (version 2015-06-18) [24]. For each project, we calculated the mean number of PRs (mean.prs) and the mean number of integrators (mean.integrators) on a per month basis for the period July 2013 to July 2014. Per metric, we split projects into three equally sized groups (small, medium and large). We also calculated whether

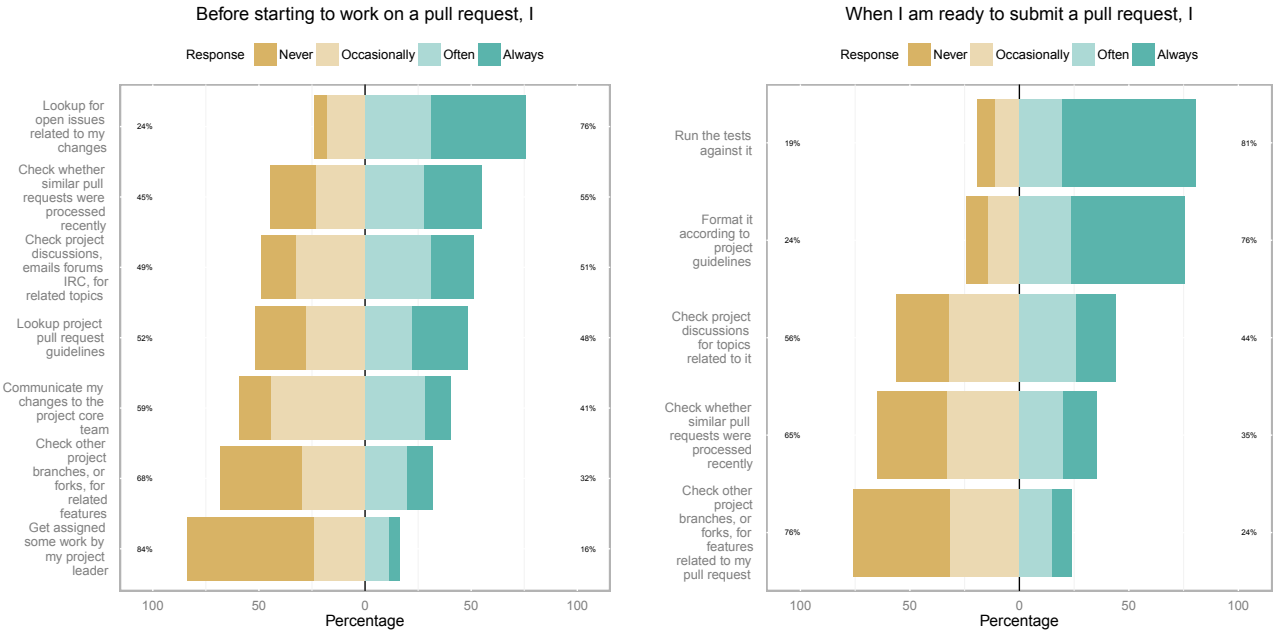

Figure 2: Work practices before (left) and after (right) coding a contribution.

respondents belong to the top 10% of contributors (top.10.perc.contrib) for the repository they reported and whether they usually commit to big, medium or large projects (typical.size.of.project).

To ensure that our answer set included developers that contribute primarily or exclusively through PRs rather than through other means (*e.g.*, bug reports), 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 77 respondents who did not indicate contributions exclusively through PRs or through branch-to-branch PRs. The final answer set contained 645 answers.

4. RESULTS

This section presents the results of our exploratory investigation. When quoting survey respondents, we refer to them using a [rX] notation, where *X* is the respondent's ID. Codes resulting from coding open-ended answers are *underlined*. When referring to quantitative results, we annotate the metrics presented in Section 3.3 with a sans-serif font. Where applicable, we integrate and compare our findings with related research findings.

RQ1.1: What are contributors' work practices?

We wanted to understand which practices contributors use after they decide to create a PR, before and after the actual coding, but before they submit it. A variety of survey questions led to this understanding and the answers are presented in Figure 2.

Work practices followed before coding.

To ask respondents about their work habits *before* coding, we provided them with a set of 7 questions (based on our analysis of the literature and our vast GitHub experience) with a 4-level Likert scale. Only 24 (3%) respondents added information using the 'other' field, mostly providing clarifications. Results show that, in general, contributors reported to conduct all the mentioned activities (as one developer put it: "*These are all reasonable things to do*" [r490]). Nevertheless, the activities receive different emphasis. In particular, contributors reported practices mostly related to increasing their *awareness* (*i.e.*, "an understanding of the activities of others, which provides a context for your own activity" [14]). They checked whether similar work had already been performed by consulting (in this order of frequency) the issue tracking system, previous PRs, project communication channels, and external branches/forks. In the 'other' field, respondents added 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]). The top experienced contributors (metric: top.10.perc.contrib) said that they do not need to update awareness because they maintain mental models of the status of the project: "*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]

Work practices followed after coding.

We asked developers to rate 5 common practices with a 4-level Likert scale. The 'other' field was filled in 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 declared they *do not* recheck whether similar work has been accomplished in the meanwhile—a contrast to what they report to do before starting to code. The activities that developers described as the most frequent before submitting a PR are formatting the code according to the project's guidelines and running tests against the completed code. Some respondents complained how there was a lack of support for these activities in 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]).

RQ1.2: How do contributors assess their code?

We sought to understand how contributors evaluate the quality of their code before submitting it as a PR. Examining the quality of

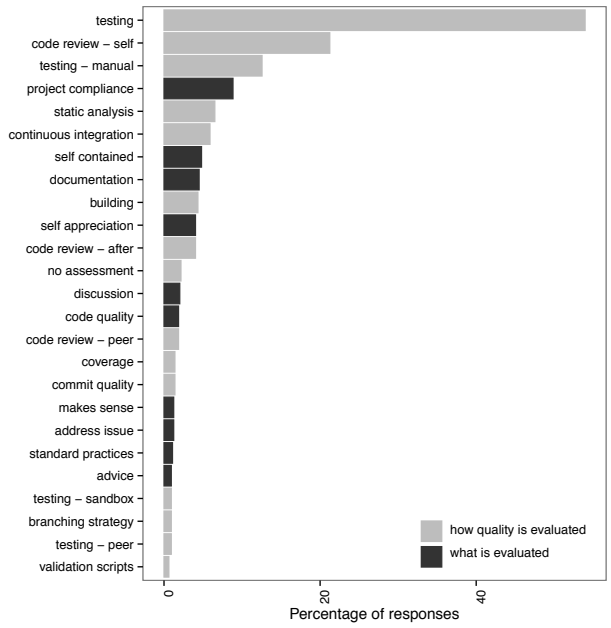

Figure 3: How contributors determine their contributions' quality.

contributions is very important for contributors; similarly, contribution quality is the number one factor integrators examined to decide whether to accept a contribution or not [27]. To investigate how contributors assess the quality of their contributions, we asked them a compulsory open-ended question. Although the question was specific on *how* contributors evaluate quality, the analysis of the results also revealed *what* contributors examine in their PRs. (Figure 3 summarizes these results.)

How quality is evaluated.

The most common mechanism that contributors use to assess the quality of their contributions is *testing*. More than 60% of the responses mentioned employing a form of automated testing (running on the developer's workstation or *continuous integration* servers). This highlights the practicality of automated testing for evaluating contributions and complements the integrator's opinion of testing as a contribution evaluation mechanism [27]. Around 10% said they also perform *manual testing* of their changes by running their contribution against the main repository, either in specialized *sandbox* environments or by their *peers* (*i.e.*, collocated colleagues [r125] or other project developers [r198]).

On the tool front, 7% of the respondents use *static analysis* tools for automatically evaluating their contributions. A wide range of these tools were reported, mostly belonging to three categories: lint 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 predefined style (*e.g.*, CHECKSTYLE), and formal method tools to detect logical inconsistencies (*e.g.*, CPPCHECK). As an extra step, contributors *build* the software to catch simple errors with the compiler. Building usually also invokes the project's test suite.

A complementary examination mechanism that contributors mentioned is *code review*; usually (20%) of it is performed by the contributors themselves before they submit a PR. During code review, the contributors examine properties in the source code and documentation as discussed above. When conditions allow, contributors ask their *peers* to do a code review before they submit a pull request; the peers may be colleagues (*e.g.*, [r57,172]) or other members of the project community [r382]. Some contributors may 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 [breaking] changes.*" [r399]). A related response (given by 4% of the respondents) mentioned that they do not explicitly assess the quality of their PR as they either believe it is the project owner's responsibility (*e.g.*, [r217,392]) or they count on immediate acceptance (*e.g.*, [r245]).

Finally, contributors examine the PRs' quality through *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]).

What is evaluated.

One of the top priorities for contributors when examining PR quality is *compliance*, which had many manifestations in our response set. The most common 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, mainly 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 (*e.g.*, "*Following clean code principle and checking code style*" [r143]).

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 the change is minimal ("*the code* [...] *contains only the minimal amount of code change necessary to implement the feature*" [r15]). Several contributors reported that they 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 PR'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 PRs to be *self contained* (*e.g.*, "*It should be focused on the feature to implement (or bug to fix). Nothing unrelated to the topic should be in there.*" [r52]) and they also try to ensure that the *documentation* of both their code and the PR eases the comprehension of the PR and meets assumed project standards (thereby enhancing compliance).

RQ1.3: How are changes communicated?

In the question about contributors' work habits before coding, we asked about how often they communicate the changes to the project core team. Most of the respondents (59%) reported they did not communicate with the core team or that they communicated with the team very occasionally (see third item from the bottom in the left-hand side of Figure 2). We analyzed the communication behavior in relation to the size of the projects (metric: mean.prs) reported by the respondents. Although we found a significant relationship ($p < 0.001$, assessed using the χ^2 with $df = 6$) between the size of the reported project and the reported frequency in communication, which goes in the direction of communicating slightly more when

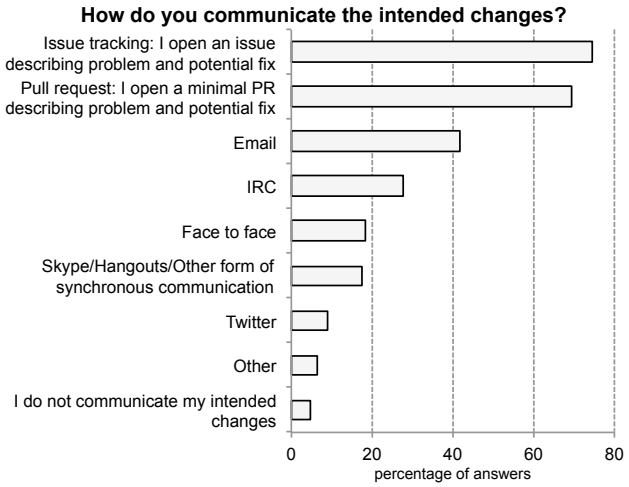

Figure 4: How contributors communicate with integrators.

projects are larger, the strength of the relationship[2] is very weak (Cramer's V = 0.14).

In a subsequent multiple choice question, we inquired about the communication means contributors use when they decide to communicate on a change. The summarized results are presented in Figure 4. Many respondents explained that they open an issue in the tracker or a new PR, or both. They use emails or more synchronous communication channels (*e.g.*, IRC or instant messaging) less frequently. 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 mailing lists, toward more structured channels, such as issue trackers.

Those that added information in the 'other' field (6%), mainly specified the communication channels they use. Many mentioned 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]), and a few reported to use email-based communication (*e.g.*, "*Mailing list to get the opinion of the community and core team*" [r286]).

RQ2: What are the challenges of contributing?

To find the pain points experienced when contributing through the pull-based model, we explicitly introduced a mandatory open-ended question in the survey and asked respondents to state the biggest challenge they faced when contributing PRs. We learned that challenges revolve 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 relates to the *code* theme (*e.g.*, "*using the project code style*" [r66]), while in others it relates to the *social* theme (*e.g.*, "*Not knowing all the rules/process*" [r62]). The results are summarized in Figure 5. From left to right, we first classify the answers on the contributor's rank (*i.e.*, whether it is in top 10% PR contributor or not), then we show the three main

[2]We obtained similar results aggregating into two groups both size (*i.e.*, 'small' and 'medium' vs. 'large') and communication frequencies (*i.e.*, 'never' and 'occasionally' vs. 'often' and 'always').

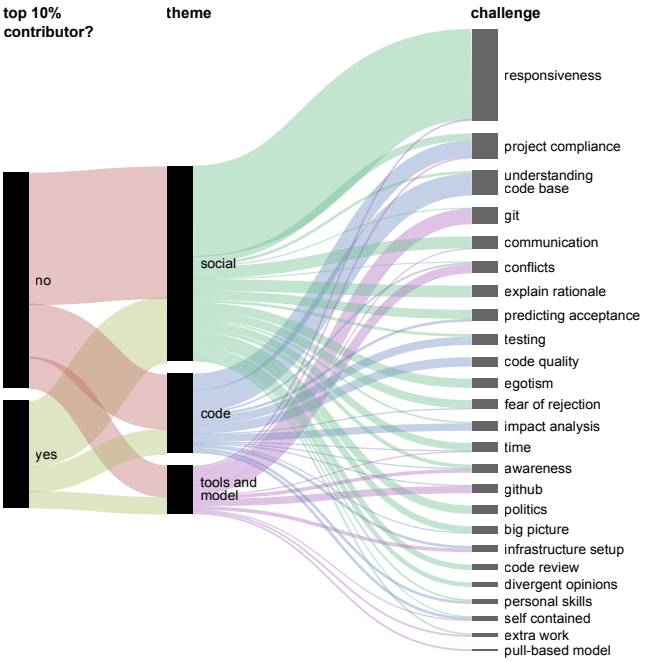

Figure 5: Challenges contributing with the pull-based model.

themes and how the answers flow into the specific challenges. The thickness of a line represents the number of responses.

We expected the top contributors (metric: top.10perc.contrib) to be less affected by *tools and model* and *code* related challenges given their greater experience, but to our surprise, both types of contributors had a very similar distribution of challenges among the three main themes. The theme reported by the majority of the respondents to be the most challenging when contributing in a pull-based model is the *social* one, followed by *code* and *tools and model*, respectively. We discuss the results in this order.

Social aspects.

The *social* theme is connected to most of the reported challenges in different ways, with the most prominent being *responsiveness*. More than 15% of the survey participants mentioned that getting timely feedback, if any, for their pull requests is hard and usually related to people issues (*e.g.*, "*The owner of the repo doesn't ever respond to the PR and leaves it hanging open forever*" [r15], "[there are] *projects with lots of open PRs and few actually being accepted*" [r98]). This situation seems to generate frustration for the contributors and they start to lose interest in the project: "*Malaise and abandonment. Few things are more frustrating than opening a PR 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 specified that they would rather receive a clear reject than have no response to 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 improved *prediction of acceptance* of future PRs. Not knowing whether a PR will be accepted poses 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* was reported as another issue. Some contributors specified that they find it challenging to *explain the rationale* of their changes, which can af-

fect 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]) reported a *fear of rejection* as they found it personally embarrassing when their work was judged to be inadequate 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 reported (*e.g.*, "*Fear of looking stupid. Fear of rude response.*" [r190], "*Discouraging project owners*" [r228]). In particular, respondents described 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 PRs 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 PR.*" [r251]).

Furthermore, contributors reported that it is 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 one is *understanding the code base* of the project, including layout and architecture (*e.g.*, "*Read others code and get understanding of the project design*" [r564]). This problem seems to be magnified by project size and a 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 also find it 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 PR 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, and avoid regression, contributors explained 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 PRs*" [r656]).

Writing PRs with proper *code quality* is mentioned as the only issue by 13 developers. Another 7% reported 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 a higher design level; this challenge also highlights the difficulties in knowing the format for PRs, commit messages, *etc.* ([r277]). Some respondents explained that this challenge is due to tribal knowledge, *i.e.*, information only known within the project team and not explicit to the outside world (*e.g.*, [r500,659]).

Tools and model.

Respondents reported challenges regarding the *tools and model* less frequently. Among those, the use 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 mentioned having problems with the local *infrastructure setup* needed for development and testing. The few explicit answers about the *pull-based development model* mostly relate to its learning curve (*e.g.*, [r572]). More respondents mentioned *GitHub* as a challenge, especially when it comes to having 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 further investigated what respondents think projects should do to reduce barriers for new contributors. This was mapped in the survey to an open- ended question that 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) deals with 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%). For the second barrier, respondents explained 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 stated 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/PRs/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, including 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, 12% of the respondents reported better code *documentation* as important to attract new contributors.

5. DISCUSSION

We now discuss our main findings, contrasting them with findings about the integrators' perspective. We suggest recommendations for practitioners and consider the implications for researchers.

5.1 Main findings

Awareness. Contributor practices for increasing awareness in the pull-based development setting are not substantially different from practices in settings where no social features are available. Similar to participants in other studies of OSS contributors [36, 38, 44], PR contributors first attempt to build an understanding of the project's current status by examining existing contributions, the project's issue database, and any contribution guidelines. What was surprising was that they did not explicitly discuss the social features [11] of GitHub as a source of information as much as (we) expected. Actually, only one respondent specifically mentioned the use of a GitHub social feature to build awareness about a project ("*I see changes in RSS channel*" [r600]). This situation raises questions about whether social features are useful to contributors. They might, however, rely on subtle social signals that environments like GitHub provide, without realizing it.

Moreover, while most contributors report that they use the issue tracker for finding similar issues or PRs, at the same time, many PRs are rejected because they are duplicate or superseded [25]. Intuitively, since contributors report that they are aware of the project status before they start working on a contribution, one would expect that very few PRs would be rejected for such reasons. This indicates a critical step between contemplating a contribution and actually creating it, and it underlines the importance of improving how awareness about projects is created and maintained.

We also noticed an interesting paradox. Contributors deem it is important to spend time checking for existing work related to the PR, but once they start coding a PR, they rarely (if ever) communicate the intended changes to the core team. The paradox is that they report it is important to be aware of what is going on in the project, but they do not express the intention to personally invest their own time to increase the overall awareness in the project.

Furthermore, the fact that contributors often prefer to use communication means other than PRs (see Figure 4) hinders awareness. Not only are multiple discussions spread across different PRs, but also communication becomes scattered over multiple channels, making it difficult, if not impossible, for new contributors to understand the rationale behind a change.

Transparency. The pull-based development model, in conjunction with the social media functions offered by GitHub, makes contributions and their authors more prominent than in other contribution models. As Dabbish *et al.* put it: "[it makes] the work visible" [12]. Indeed, various developers discuss whether [1, 50] or not [8] GitHub profiles should be the new *de facto* CV for developers. In our survey, we included an additional question asking *why* participants contribute to the chosen project. We introduced a closed question with 7 non-mutually exclusive answer options (based on our analysis of the related literature) and an open text field to specify 'other' reasons. While most answers validated well-known motivations for contributing to OSS (*e.g.*, the main motivation (60% of the respondents) was 'usage' of the project they contribute to), approximately 35% of the respondents explained that they contributed for reasons related to personal career development, while 23% of the respondents mentioned enrichment of their public profile/CV as a motivating factor (*e.g.*, "*Making contributions to* [project] *makes it easier for me to get new clients*" [r121]). Lerner and Tirole formalized that contributions to OSS projects are also driven by a *career concern* incentive. This incentive increases in strength as the audience's visibility into the performance increases [33]. Some of our respondents also indicated that contributions to GitHub benefit their career growth (e.g., "*My contribution to* [projects] *allowed me to obtain a job within my favorite subjects*" [r437].

Additionally, several contributors fear that rejection of their PRs may harm their reputation. If this fear of rejection is caused by transparency, as also suggested in the previous analysis by Dabbish *et al.* [12], we have additional quantitative evidence about its benefits but also about potential risks.

Responsiveness. More than 100 participants complained about the poor responsiveness from integrators. Specifically, they reported that they were worried they would not get a response or that they would get it too late to be relevant. They also suggested that improving the speed of responding to a PR would be an effective way to reduce barriers for newcomers. This complements the findings reported by Zhou and Mockus on an analysis of the ecosystems of Mozilla and GNOME [54]. Zhou and Mockus found that "low attention [...] as evidenced by a too-rapid response to issue reports" reduces the chances of a newcomer becoming a long-term contributor. We note that our data is drawn from self-reported behavior and suggestions given by contributors, while the study of Zhou and Mockus was mostly performed on traces left on software repositories, and therefore, the different development settings may lead to different ways of tuning out unwanted contributions.

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 pervasive concern for both contributors and integrators and its effects are usually detrimental. Asynchrony hinders the observability of the overall status of a project and burdens integrators and contributors with extra communication obligations. Recently, several high profile companies (*e.g.*, Facebook [22] and Google [32]) have moved away from the pull-based model, while others use strictly bounded code review processes and branching strategies (*e.g.*, Microsoft [5, 2]) to increase development speed for their internal repositories (however, they still use pull-based development for OSS projects).

From a distributed systems theoretic standpoint, mitigating the results of asynchrony is impossible [42]. Therefore, integrators and contributors should agree on minimal communication protocols that increase each other's awareness and rendezvous points for mandatory information exchange. In certain cases (*e.g.*, collocated development), projects should be prepared to abandon the pull-based model in favor of more direct feedback loops.

5.2 It takes two to tango

The pull-based development mechanism, and its GitHub implementation in particular, aims to facilitate the information exchange between two interacting parties sharing a common goal, namely integrating a change into an existing code base. Due to the closeness and asynchrony of the interaction, it is expected that good or bad practices of one interacting part will reflect on the other. Comparing this research with our previous work [27], we found a number of technical and social pain points experienced by both integrators and contributors. We report on these below.

Quality. Contribution quality is a major concern for contributors. It is one of the most frequently reported challenge items and also something they deeply care about before PR submission. Not surprisingly, quality is also a top priority for integrators. A cross examination of the factors that contributors and integrators examine in PRs reveals that there is also a high overlap in terms of compliance/conformance and code quality as top factors. Moreover, automated testing is used by both integrators and contributors as a commonly accepted way to ensure contribution quality. We hypothesize that this shared understanding of quality, and the ways of achieving it, is the result of widely accepted technical norms. Positively, this helps the majority of contributions to be accepted (85%), while rejections are usually not due to technical reasons [25].

Lack of process. The pull-based model on GitHub lacks a specific patch acceptance process (as is the case with Gerrit [40] or CodeFlow [2]). Some integrators find the lack of a well-defined acceptance process (e.g. voting and sign-off) disturbing enough to move to other reviewing platforms. In addition, experienced contributors are used to searching for PR process documents, though such policy documents are often not present in smaller projects. A GitHub-wide acceptance process definition and enforcement mechanism might be beneficial to both integrators and contributors and might deter one-off, low-quality contributions. More research is needed to explore options concerning process policies.

Workload and responsiveness. Integrators on large, active projects reported that they have problems handling and prioritizing the large number of PRs those projects attract; perhaps as a result, contributors complained about the lack of responsiveness. However, integrators also protested about the lack of responsiveness from con-

tributors when they request additional changes during a code review and complained about "hit-and-run" PRs. These concerns may be the result of the pull-based development model that simplifies experimentation with contributions to projects without reducing the reviewing burden on both parties.

Communication tooling. A significant portion of both integrators and contributors find that the communication facilities afforded by the GitHub PR mechanism are lacking in terms of immediacy and structure. This hinders the effective discussion of high-level concerns (*e.g.*, system design) and has a negative impact on the centralization of information about a contribution. Indeed, many contributors and integrators reported that they use external tools, mainly supporting synchronous communication (*e.g.*, IRC or instant messaging), to exchange information. Two key features that are missing, as reported from our respondents, are threaded communications and voting mechanisms.

Communication failures. Communicating about the rationale for PRs was reported as difficult not only by contributors, but also by integrators wishing to understand the reasons for a PR. Integrators complained that discussions on PRs diverge from technical content, while contributors expressed their concerns about having to cope with project politics in order to get their contributions accepted. Contribution rejection is a concern for both parties: integrators reported that it is not easy to explain the rational behind a rejection, while contributors stated that it is hard to accept the rejection. We conjecture that the above shared difficulties are the result of a communication process that, while open and accessible, is lacking in terms of immediacy and traceability.

5.3 Recommendations for practitioners

We present a set of recommendations that can help streamline the experience for contributors when working with integrators in the pull-based model. For one-off contributions, these guidelines revolve around two basic principles: *minimizing friction* and *maximizing awareness*. For more *long-term involvement* in a project, it is crucial to build and maintain a contributor profile.

Minimizing friction. Contributions that are *small* and *isolated* are easier for integrators to process. In previous work [51, 3, 25, 27], the size of the change was one of the most important factors related to acceptance. This is 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.

Projects should provide a policy or *comprehensive set of contribution guidelines*. These guidelines should at least provide details about the expected code style, commit format, PR process, and available communication options. Well-thought- out guidelines will help developers format contributions using the expected style and can act as a reference in code review discussions. Moreover, projects should *invest in good tests*. Not only would contributors gain confidence about their contributions by testing them locally, but by doing so, integrators will evaluate them more quickly [25].

Automation is also important and it should at least cover the *development environment setup*. Ideally, the contributor should be able to set up a fully working development environment by running a simple command; existing tools allow this (*e.g.*, Vagrant and Ansible). Projects should also invest time to set up *automatic quality evaluation* of incoming contributions. This can include code style compliance checks and perhaps more sophisticated static analysis tools. In the case of GitHub, external services are available to enable continuous integration (*e.g.*, Travis) and code quality (*e.g.*, Code Climate) monitoring on a per contribution basis.

Maximizing awareness. Awareness can be increased by contacting the development team using real-time communication channels (*e.g.*, IRC or its evolved counterpart GITTER, which is better integrated in GitHub) or by following the minimal PR idiom [7] (depending on project preferences). Integrators should be both *proactive*, by establishing (and perhaps even documenting) a professional communication etiquette, and *reactive*, by following discussions and intervening in cases where discussion diverges from the etiquette. Similarly, contributors should be available after a submission to promptly discuss the results of the code review and thus mitigate some of the negative effects of asynchrony.

Long-term involvement. For contributors seeking long-term involvement in project communities, essential steps are *profile building* through a stream of excellent contributions and *participation* in other community activities (*e.g.*, discussion of issues); integrators both evaluate [34, 27] and prioritize [27] work using a mixture of social signals and developer track records, whose visibility is ensured by the transparency of the model.

5.4 Implications for researchers

Our work uncovers several future research directions.

Work prioritization. Low responsiveness is one of the most recurring challenges experienced by contributors. Integrators also reported problems in prioritizing PRs [27]. Automating the prioritization of PRs could help integrators allocate their time more effectively but also 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 a potentially automatically generated guideline. Initial work in this direction has been carried out by van der Veen *et al.* [47].

Estimated time for merging. A widespread usability heuristic states that a "system should always keep users informed about what is going on" [37]. This is often achieved, for example, through progress bars in application UIs. Contributors' frustration about 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 an indication on the most significant factors considered for the prediction, contributors could take advantage of this prediction to understand what could be improved to speed up acceptance (*e.g.*, splitting a PR into self-contained tasks), which would also help contributors speed up their decision on whether to continue contributing to a particular project. Previous research has estimated merging time for patches [30], closing time for issue reports [19, 53], *etc.* In the context of PRs, Gousios *et al.* [25] developed a machine learning approach to predict a pull request's merge window. In addition, Vasilescu *et al.* [48] developed models to determine PR processing time. This is a ripe opportunity for researchers to support a wide population of developers.

Untangling code changes. Integrators reported that code understanding and reviewing is simplified if code changes pertain to a single, self-contained task [2, 27], however, contributors reported that creating them is a challenge. Recently, researchers have proposed automated approaches to split changes into self-contained tasks [29, 13]. It is an interesting opportunity to apply these methods and integrate them into 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 PR interface as an optional service.

Improved awareness and communication. Our respondents reported the need to build awareness before working on a new PR, but expressed little intent to communicate changes to the core project team before starting work. Understanding this phenomenon is an interesting avenue for further research on collaboration behavior in knowledge-intensive settings. Moreover, a number of drawbacks emerged in communication occurring within PRs, despite the advantage of being close to the changed code. Particularly, communication support should be improved for discussing high-level concerns and for scaling to longer discussions. Multidisciplinary studies involving user interface designers, communication experts, and software engineering can be designed and carried out to determine how to improve communication within PRs.

5.5 Design implications

The pull request model offers a simple, yet solid basis for distributed collaboration. The lowered barrier to entry, transparency of social platforms, and integration of both analysis tools and reviewing mechanisms help projects expand their collaborator base seamlessly. Considering their continuous growth in popularity, it is reasonable to expect that pull requests will become the minimum unit of software change in most collaborative projects. Our current and previous findings hint at the design of features that will facilitate this transition. We imagine a contribution platform, which we call PR.next, that optimizes the contribution experience and helps integrators handle the reviewing load by means of intelligent algorithms, which we describe in the following.

Initially, PR.next assists contributors in evaluating their contribution proposals against the state of the project. A contributor expresses the proposed change in natural language and the system searches i) the code base and ii) open or recently closed contributions for similar changes. Then, PR.next helps contributors format their contributions by running continuous integration and style checks in a *private staging space* before the change becomes public. In the mean time, PR.next compares the contributions to other in-flight contributions and warns about duplicates. PR.next also helps integrators prioritize their work. When a new contribution arrives, PR.next combines information from multiple analysis tools (*e.g.*, continuous integration) to rank pull requests according to their readiness to be reviewed. The system gives integrators visual hints (*e.g.*, predicted time to merge) and mines information, such as discussion status or voting results, from other in-flight contributions to help them evaluate priority. At the project level, PR.next supports community voting mechanisms to help projects evaluate contribution desirability, where votes are ranked for importance based on voter characteristics in the social (voter status in community) or dependency (voter's project status within the software ecosystem) graph.

6. LIMITATIONS

We designed our survey with the stated aim of gaining insights on a novel mechanism of collaboration in distributed software development. For closed selection questions in the survey, the response categories originated from our review of the literature and also from our prior experience working with and researching [25, 27] PRs and the pull-based model. The questions were phrased to avoid leading the respondent to a specific answer and were validated through (1) consultation with colleagues expert in qualitative research, (2) a formal pilot run, and (3) several mini-runs of the survey. Despite our best efforts, there could be several reasons why our study is limited.

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. To alleviate this issue while coding, we allowed more than one code to be assigned to the same answer. Question-order effect [43] (*e.g.*, one question could have provided context for the next one) may lead the respondents to a specific answer. One approach to mitigate this bias could have been to randomize the order of questions. In our case, we decided to order the questions based on the natural sequence of actions to help respondents recall and understand the context of the questions asked. Social desirability bias [18] (*i.e.*, a respondent's possible tendency to appear in a positive light, such as by showing they are fair or rational) may have influenced the answers. To mitigate this issue, we informed participants that the responses would be anonymous and evaluated in a statistical form.

Generalizability – Transferability. Our selection of projects and contributors to GitHub projects using the pull-based model may not be indicative of the *average* project. Previous work [25] found that the median number of PRs 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 may be different. This may affect the results on obstacles such as low responsiveness and inefficient communication, as average projects do not use PRs in a high capacity. To reduce other limitations to the 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 social coding sites featuring the pull-based development model and it has specific social media features. While this model remains the same across all these sites and the social features are similar, 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 witnessed by the fact that many open-ended answers included direct references to GitHub or tools in its ecosystem (*e.g.*, Travis CI).

7. CONCLUSIONS

We presented our investigation of the pull-based development model as implemented in GitHub from the contributors' perspective. Our goal was to gain knowledge on the work practices of pull request contributors and the challenges they face. We make the following key contributions:

(1) A publicly available [26] iteratively-tested survey with questions for eliciting contributors' practices in the pull-based model, and the anonymized answers of 760 respondents.

(2) The set of open-ended questions we coded manually, and the R analysis scripts for the overall data analysis.

(3) A thorough analysis of the answers to our research questions on contributors' work habits, PR preparation, and open challenges in contributing with the pull-based model.

(4) A discussion comparing our findings with previous literature, recommendations for practitioners using the pull-based model, and data-derived directions for future research and design.

Among our findings, we identified reducing response time, maintaining awareness, improving communication both in content and in form, and quality assessment as key components for supporting contributions in the pull-based model. We hope that our insights will lead to merging external contributions more effectively in practice and to devise improved tools, to support developers in both creating and handling code contributions more efficiently.

8. REFERENCES

[1] GitHub is your new resume. https://news.ycombinator.com/item?id=2763182. Accessed 2016/02/15.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] Bitbucket. http://bitbucket.org/. Accessed 2016/02/15.

[7] B. Bleikamp. How we use pull requests to build GitHub. https://github.com/blog/1124-how-we-use-pull-requests-to-build-github. Accessed 2016/02/15.

[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 2016/02/15.

[9] J. M. Corbin and A. Strauss. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative Sociology*, 13(1):3–21, 1990.

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

[11] 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.

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

[13] 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, pages 341–350. IEEE Computer Society, 2015.

[14] 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.

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

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

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

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

[19] 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.

[20] GitHub. https://github.com/. Accessed 2016/02/15.

[21] Gitorious. http://gitorious.org/. Accessed 2016/02/15.

[22] D. Goode and S. Agarwal. https://code.facebook.com/posts/218678814984400/scaling-mercurial-at-facebook/. Accessed 2016/02/15.

[23] G. Gousios. http://ghtorrent.org/pullreq-perf/. Accessed 2016/02/15.

[24] 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.

[25] 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.

[26] G. Gousios, M.-A. Storey, and A. Bacchelli. Pull request contributors analysis dataset. http://dx.doi.org/10.5281/zenodo.46063, Feb. 2016.

[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 2016/02/15.

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

[34] 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.

[35] 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.

[36] 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.

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

[38] 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.

[39] T. Punter, M. Ciolkowski, B. Freimut, and I. John. Conducting on-line surveys in software engineering. In *International Symposium on Empirical Software Engineering*, ISESE'03, pages 80–88. IEEE, 2003.

[40] 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.

[41] 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.

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

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

[44] 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 1379–1392, 2015.

[45] 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.

[46] 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.

[47] E. van der Veen, G. Gousios, and A. Zaidman. Automatically prioritizing pull requests. In *Proceedings of the 2015 International Working Conference on Mining Software Repositories*, pages 357–361, 2015.

[48] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov. Quality and productivity outcomes relating to continuous integration in GitHub. In *10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, ESEC/FSE, page accepted. IEEE, 2015.

[49] 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.

[50] B. Weiss. GitHub is your resume now. http://anti-pattern.com/github-is-your-resume-now, June 2012. Accessed 2016/02/15.

[51] 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.

[52] 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.

[53] 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.

[54] M. Zhou and A. Mockus. Who will stay in the FLOSS community? modeling participant's initial behavior. *IEEE Transactions on Software Engineering*, 41(1):82–99, 2015.