Almost There: A Study on Quasi-Contributors in Open Source Software Projects

Igor Steinmacher
Federal University of Technology, Paraná
Campo Mourão, PR
Northern Arizona University
Flagstaff, AZ
igorfs@utfpr.edu.br

Gustavo Pinto
Federal University of Pará
Belém, PA
gpinto@ufpa.br

Igor Scaliante Wiese
Federal University of Technology, Paraná
Campo Mourão, PR
igor@utfpr.edu.br

Marco A. Gerosa
Northern Arizona University
Flagstaff, AZ
marco.gerosa@nau.edu

ABSTRACT

Despite the importance of open source software (OSS), recent studies suggest that well-known OSS projects struggle to find the needed workforce to continue evolving In this paper, we investigate how and why *quasi-contributors* (external developers who did not succeed in getting their contributions accepted to an OSS project) fail. To achieve our goal, we collected data from 21 popular, non-trivial GitHub projects, identified quasi-contributors, and analyzed their pull-requests. In addition, we conducted two surveys with (1) the identified quasi-contributors and (2) projects' integrators to understand their perceptions about nonacceptance. We found 10,099 quasi-contributors — about 70% of the total actual contributors — that submitted 12,367 nonaccepted pull-requests. In five analyzed projects, we found more quasi-contributors than actual contributors. About one-third of the developers who took our survey disagreed with the nonacceptance, and around 30% declared the nonacceptance demotivated or prevented them from placing another pull-request. The main reasons for pull-request nonacceptance from the quasi-contributors' perspective were "superseded/duplicated pull-request" and "mismatch between developer's and team's vision/opinion." A manual analysis of a representative sample of 263 pull-requests corroborated with this finding. We also found reasons related to the relationship with the community and lack of experience or commitment from the quasi-contributors. This empirical study is particularly relevant to those interested in fostering developers' participation and retention in OSS communities.

CCS CONCEPTS

• **Software and its engineering → Open source model**; • **Human-centered computing →** *Collaborative and social computing*;

ICSE, 2018, Sweden
© 2018 Copyright held by the owner/author(s).
ACM ISBN . . . $00.00
https://doi.org/

KEYWORDS

pull-requests, quasi-contributors, newcomers, open source software

ACM Reference Format:
Igor Steinmacher, Gustavo Pinto, Igor Scaliante Wiese, and Marco A. Gerosa. 2018. Almost There: A Study on Quasi-Contributors in Open Source Software Projects. In *Proceedings of International Conference on Software Engineering, Sweden, 2018 (ICSE),* 11 pages.
https://doi.org/

1 INTRODUCTION

Sustaining work skills and knowledge in large, complex projects is a formidable undertaking yet crucial for the maintenance of long-lived OSS projects [5]. However, the number of developers willing to compromise to this often non-paid activity is not large. As a result, 64% of well-known, nontrivial, and popular OSS projects rely on 1–2 contributors to manage most of their tasks [1]. Drop off from contributors is one of the main problems that OSS projects face [3, 12]. Conversely, many developers, newcomers to a project, send contributions which are not incorporated into the source code and give up trying. Such "getting contributions accepted" barrier is frequently reported as a major reason raised by newcomers that dropped out of the projects [26, 27].

Little is known about these quasi-contributors, i.e., developers who have only nonaccepted contributions to a given OSS project. The literature focus on factors that affect the likelihood to accept a contribution [8, 11, 13, 18, 22, 30, 31] or how long it may take [32], analyzing indistinctly non-accepted contributions from actual[1] and quasi-contributors. In this study, we focus only on the quasi-contributors side.

We analyze projects that follow the pull-request model, since it is believed to offer a low barrier to entry for potential contributors [35, 36] and has been defined as the default model in well-known social coding websites, such as GitHub and GitLab [9]. According to McDonald and Goggins [16], the pull-request model leads to greater participation and more opportunities for review and feedback from the community, as well as greater visibility for potential contributors.

[1] We use the term "actual" to refer to contributors that have at least one accepted pull-request in the project.

In this paper, we investigate how common are quasi-contributors (and quasi-contributions) in OSS projects that adopt the pull-request model (**RQ1**), what are the reasons for nonacceptance (**RQ2**), and how quasi-contributors perceive the nonacceptance (**RQ3**). To achieve these goals, we collected and analyzed data from 21 active, nontrivial, and popular OSS projects. In addition, we conducted two surveys: one with 355 quasi-contributors; and another with 21 project integrators, (*i.e.,* developers who are in charge of accepting contributions [10]). By analyzing this data, we make the following contributions: (i) we bring the attention to one relevant yet neglected OSS developer, the quasi-contributor; (ii) we provide evidence that quasi-contributors are rather common in popular OSS projects — they are about 70% the number of actual-contributors; (iii) we elucidate 19 reasons regarding nonacceptance through a survey with quasi-contributors; (iv) we corroborate these findings by cross-validating the results with an additional survey with 21 projects' integrators, and with a manual inspection of a representative sample of 263 quasi-contributions; (v) we show that nonacceptance might incur in demotivation, potentially preventing further contributions.

2 RELATED WORK

Pull-based development is an emerging paradigm for distributed software development that has been attracting open and closed source projects, which are migrating their code to this environment [4, 8]. According to Gousios *et al.* [8] in pull-based development, the work is distributed between a team, which submits, often occasionally [19], changes to be considered for merging, and a core team, which oversees the merging process, provides feedback, conducts tests, requests changes, and finally accepts the contributions.

Previous research has studied pull-based development; for example, Gousios *et al.* [8] found that the decision to merge a pull-request is related to recently modified the code. They also found that 53% of non-merged pull-requests are unmerged for reasons related to the distributed nature of pull-based development, and only 13% of the pull-requests are unmerged for technical reasons. They report that the decision to merge a pull-request is mainly influenced by whether the pull-request includes recently modified code and that the time to merge is influenced by the developer's previous track record, the size of the project and its test coverage, and the project's openness to external contributions. Padhye *et al.* [18] complement this list, reporting that bug fixes are more likely to be merged into the main projects than feature enhancements. While these studies analyzed the reasons behind contribution nonacceptance, in this paper, we complement this literature by focusing on contributors whose pull-requests were not accepted.

Tsay *et al.* [31] focused on understanding the relationship between socio-technical aspects and the likelihood for pull-request acceptance, reporting that the social connection between the submitter and project manager matters when the core team member is evaluating the pull-request. Furthermore, highly discussed pull-requests were much less likely to be accepted; however, the submitter's prior interaction in the project moderated this effect. Tao *et al.* [30] also found that bad timing of patch submission and a lack of communication with team members can lead to a negative review.

Gousios *et al.* also surveyed developers who merge pull-requests, the so-called integrators [10], and developers who were trying to submit a pull-request [7], called contributors. From the integrator's perspective, the authors reported social challenges that needed to be addressed, for example, how to motivate contributors to keep working on the project and how to explain the reasons for nonacceptance without discouraging them. From the contributor's perspective, they found that it is important to reduce response time, maintain awareness, and improve communication. They also found that integrators decide to accept a contribution after analyzing source code quality, code style, documentation, granularity, and adherence to project conventions. Hellendoorn *et al.* [11] also confirm that code style is an important aspect. A second signal that the integrators examine is whether the proposed pull-request is in line with the project's goals and target [10], which may be difficult for a newcomer to ascertain. On the other hand, integrators report that there is no difference in treatment of pull-requests from the core team versus those from the community and that they postpone the decision to merge in the case of technical factors. Rigby and Storey [22] also observed this postponing effect.

Rahman and Roy [20] compared successful and unsuccessful pull-requests made to GitHub projects. The authors report that a few technical problems recur in the pull-requests; when they are not properly solved, the pull-requests are not accepted. They also found differences in the acceptance ratio when they compare projects grouped by programming languages and domains. Additionally, the authors show that the failure rate of pull-requests rapidly increases when a large number of forks are created and that the number and experience level of developers involved in a project affect the success and failure rates of pull-requests. Finally, Yu *et al.* [35] used a regression model on data extracted from GitHub projects and found that several factors influence the pull-request latency, including pull-request size, project age, team size, and delay to the first human response. Soares *et al.* [25] also found similar results using association rules.

Our paper differs from previous research in that we focused on understanding "quasi-contributors" and their non-accepted contributions, understanding how often quasi-contributors try to become contributors, and why they fail. Our study is particularly relevant because newcomers face many challenges when they attempt to make their first contribution [28], and understanding the problems related to "quasi-contributors" can help researchers and practitioners think about new guidelines and tools to retain developers attempting to contribute to software projects.

3 METHOD

In this section, we state our research questions (Section 3.1), describe how we selected the open-source projects under investigation (Section 3.2), discuss how we identified the quasi-contributors (Section 3.3), explain how we conducted our surveys (Section 3.4), and, finally, how we analyzed our data (Section 3.5).

3.1 Research Questions

The overreaching goal of this study is to gain an in-depth understanding of quasi-contributors. We designed the following three research questions to guide our research:

RQ1. How common are quasi-contributors and quasi-contributions?

This research question investigates how common quasi-contributors (and quasi-contributions) are in our set of selected projects. This is an important direction that deserves investigation because, if several quasi-contributions have attempted multiple contributions and encountered nonacceptance each time, this might indicate that the open-source project is unfriendly to external contributors [4] or that its coding standard is too hard to meet. To answer this question, after identifying our target projects, we semi-automatically studied their commit logs and pull-requests. To complement our overview of quasi-contributions, for each analyzed project, we statistically studied the differences between unmerged and merged pull-requests, by analyzing the number of comments made on the pull-request, review comments (made on the commits), commits, changed files, and line of codes added and deleted.

RQ2. Why were the quasi-contributions not accepted?

To provide answer to this research question and the next, we conducted two surveys (details at Section 3.4) aimed at understanding the reasons for and perceptions about nonacceptance. We received, respectively, 335 and 21 answers to our surveys, which were quantitatively and qualitatively analyzed. We also studied a representative sample of 263 quasi-contributions, to cross-validate the survey results with the results from the repositories.

RQ3. How do quasi-contributors perceive nonacceptance?

Since open-source software is mainly driven by a community of volunteers, motivation is crucial to keep developers contributing. In our final research question, we analyze a subset of questions in our survey to understand (1) whether quasi-contributors agree with the decision to not merge their contributions; (2) whether nonacceptance incurs in demotivation to further contribute; or (3) whether the feedback received from the community was constructive.

3.2 Selecting Open-Source Projects

We selected 21 of the most popular (in terms of stars) open-source projects hosted on GitHub (excluding non-software projects, such as textbooks or bookmarks). However, when analyzing the most popular projects, we found that most of them were written in JavaScript. To foster diversity in our dataset [17], we hand-picked 5 additional open-source projects written in other programming languages and previously studied in another context [4].

Table 1 lists the selected projects and describes quantitative data (in terms of lines of code, number of commits, number of pull-requests, and number of contributors) about them. The selected projects are relevant because, at the time of data collection (March, 2017), they were:

Active Along their life-cycle, they received more than 20K pull-requests submitted by more than 14K contributors. On average, the projects received their first pull-request five years before our analysis. The projects in our data set received 15 pull-requests per month on average (min=3; max=56).

Table 1: Quantitative data about our selected projects. Lines of Code (LoC) comprises blank and non-blank lines, and it was collected from openhub.net. Project `spring` is an acronym for `spring-framework`.

Projects	Main PL	# LoC	# Commits	# PR	# Contributors
angular	TypeScript	472k	7K	5,649	1,582
bitcoin	C++	166k	13K	6,776	428
bootstrap	JavaScript	48.5K	15K	7,203	844
caffe	C++	73.9k	4K	1,620	233
d3	JavaScript	41.5K	4K	1,050	119
django	Python	236K	24K	8,097	1,379
docker	Go	194K	31K	16,690	1,642
flask	Python	9.6K	2K	991	381
jenkins	JavaScript	108K	24K	2,694	455
joomla!	PHP	368K	28K	10,127	535
jquery	JavaScript	45.4K	6K	2,222	260
kubernetes	Go	142K	45K	24,893	1,113
laravel	PHP	75.2K	5K	2,533	394
mongo	C++	114K	37K	1,121	281
opencv	C++	161K	20K	6,065	696
rails	Ruby	245K	60K	17,872	3,258
react	JavaScript	104K	8K	4,815	956
redis	C	142K	6K	973	220
scikit-learn	Python	191K	21K	4,317	818
spring	Java	648K	14K	1,205	201
tensorflow	C++	738K	15K	3,015	716

Popular On GitHub, the number of stars is a proxy for popularity [2]. The average number of stars is 24K (min=2.2K; max=107.K). The median number of forks is 10.1K (min=2.4K; max=81K).

Non-Trivial They have, on average, 7 years of historical records. Most of them are written in more than one programming language. They have an average of 318k lines of code (3rd quartile: 420k).

Diverse The domain of our selected projects include web-mvc frameworks (*e.g.*, `django` and `rails`), web toolkits (*e.g.*, `jquery` and `bootstrap`), data science frameworks (*e.g.*, `tensorflow` and `scikit-learn`), content management systems (*e.g.*, `joomla!`), databases (*e.g.*, `mongo` and `redis`), among others.

Frequently-Studied Some of the selected projects have been the target of several software engineering studies, such as `rails` [4, 19], `django` [1, 21], `d3` [2, 21], and `bootstrap` [1, 2].

3.3 Identifying Quasi-Contributors

After curating our selection of open-source projects, we aimed to identify the quasi-contributors. We consider quasi-contributors those newcomers to a project who submitted pull-request(s), but had no "accepted contribution" to that specific project. We consider an accepted contribution any changes that passed the pull-request cycle and, therefore, were merged to the project code base.

Pull-requests can either be open and closed, and merged or unmerged. In a first step, we selected pull-requests that were both closed and unmerged, since this indicates that the code review cycle was complete (pull-request closed) and the contribution was not accepted (pull-request unmerged). Then, for each one of the closed and unmerged pull-requests, we analyzed whether the contributor that proposed the pull-request under investigation had already provided any other pull-requests. When conducting this analysis, we observed that some contributors provided a significant number of old pull-requests (*e.g.,* before 2012) that were not classified as merged. Digging into these old pull-requests revealed that, although they appeared as not merged, some of the commits under the pull-requests were indeed merged — but through the Command Line Interface (CLI). This happened because GitHub started offering merging facilities through its web interface later 2011[2]. Before that, integrators had to merge pull-requests through `git` facilities. Therefore, to exclude these false-positive unmerged pull-requests, we restricted our search for pull-requests created after July 1st, 2011. We download our pull-request dataset on February 20th, 2017.

Moreover, we investigated whether the contributor commits had been directly merged to the main code base (*i.e.*, without using the merge button on GitHub). To do so, we locally performed a `git log` using Secure Hash Algorithms (SHA) identifier for every commit that was part of the previously identified pull-requests. The `git log` facility searches for any commit mapped to the given SHA code. If we found a commit using that SHA, we removed the developer, as well as his/her pull-requests, from our list. However, after conducting this process, some survey respondents still mentioned that their pull-requests (that our procedure categorized as unmerged) were indeed merged. As they reported, projects such as `angular.js` and `django` have specific merging procedures and, therefore, do not leverage GitHub merge facilities. As one respondent mentioned: "*My pull-request wasn't merged because my commit was rebased and fast-forward merged into the project's master branch. Some repositories only merge changes in this way, in order to avoid merging commits. AngularJS and Django are examples, as you can see in their commits history*." Consequently, we conducted another final filtering step to remove rebased commits. To do so, we queried the GitHub API using the name of the project and the usernames in our list to check whether any of the usernames in our list authored a commit under the name of another committer with a different SHA. This process also discards commits that were performed to the main codebase without pull-requests.

At the end of this process, we identified a total of 10,099 quasi-contributors who attempted to contribute a total of 12,367 pull-requests scattered across 78,381 commits. For the manual analysis, we sampled 263 pull-requests, which provides us a confidence level of 95% with a ±6% confidence interval.

3.4 Understanding Quasi-Contributors

We conducted two surveys to better understand quasi-contributors' motivations, benefits, and the problems they face. In our first survey, our target population comprised 5,138 quasi-contributors who made their valid e-mail addresses publicly available. Our second survey was delivered to 282 integrators with valid email addresses.

The surveys were based on the recommendations of Kitchenham *et al.* [15]. We also employed principles for increasing survey participation [24], such as sending personalized invitations, allowing participants to remain completely anonymous, and asking closed and direct questions. Participation was voluntary and the estimated time to complete each survey was 5-10 minutes. We obtained 335 responses (6.5% response rate) in our first survey and 21 (7.44% response rate) in our second survey. Our first survey had ten questions:

Q1. How often do you contribute to Free/Open-Source Software projects? Choices: {Daily, Weekly, Monthly, etc.}
Q2. Are you used to make contributions to different OSS projects? Choices: {Yes, No}
Q3. Have we correctly identified you as someone who attempted to contribute but did not succeed? If not, why?
Q4. What motivated you to make the pull-request(s) we identified?
Q5. Why your pull-request was not merged?
Q6. Do you agree with your pull-request being Unmerged? If not, why?
Q7. Did the unmerged pull-request prevent or demotivate you to provide more pull-requests? Choices: {Yes, No}
Q8. Were the comments from the project owners constructive? If not, why not?
Q9. How frequently do you submit a pull-request to different Projects? Choices: {Daily, Weekly, Monthly, etc.}
Q10. How often are your pull-requests unmerged (regardless of the project)? Choices: {I don't remember, None, A small part, Half, Most}

When analyzing the quantitative data from the survey, we observed that 79.1% of our respondents frequently make contributions to OSS, and 59.1% of them contribute to different projects at least bi-monthly. Interestingly, a non-negligible amount of developers (32.4%) disagreed with the nonacceptance of their pull-request. We also found that 44 developers (13.1%) informed us that we did not correctly identify them as quasi-contributors. We provide discussions to this fact in Section 7.

Q1. What is the acceptance rate of pull-requests in your project (regardless if contributed from external or internal members)? Choices: {10%, ..., 90%, I don't know}
Q2. In your opinion, what are the challenges related to pull-requests nonacceptance?
Q3. What are the common reasons for pull-request nonacceptance? Would you point some pull-request that exemplify any of the reasons?
Q4. Does your project welcome contributions from external developers (e.g., developers that are not active contributors)? If yes, how?
Q5. How does your project guide developers towards having their contributions accepted?
Q6. Are you aware of any pull-request that was not accepted due to personal or social reasons? If so, why.

In this second survey, 52.3% of the integrators thought that their projects accept 70%+ of the submitted pull-requests (33.3% had no idea). Still, 100% of the integrators said that they are willing to receive changes from external contributors, and they do so by (1) being as kind as possible, (2) introducing how-to contribute guide, and (3) welcoming pull-requests from external members (which is not always the case of OSS projects [4]). Only one integrator mentioned that PRs are not accepted "if conduct violations".

[2] https://github.com/blog/843-the-merge-button

3.5 Analyzing Data

We conducted different forms of data analysis. First, we examined the distributions of the quasi-contributions and quasi-contributors to answer our first research question (**RQ1**). For statistics, we used the non-parametric Mann-Whitney-Wilcoxon (MWW) test [33] to test whether there were differences among metrics (*e.g*, number of comments, commits, lines added and deleted, changed files, and review comments) collected from pull-requests merged and unmerged. We also used Cliff's Delta statistic, a nonparametric effect size measure that quantifies the amount of difference between these groups of observations beyond p-value interpretation. According to Romano *et al.* [23], the magnitude of delta *d* is assessed using the following thresholds: *d* <0.147 "negligible", *d* <0.33 "small", *d* <0.474 "medium", otherwise "large".

In the second analysis, we followed open-coding and axial-coding procedures [29] to qualitatively analyze open-ended questions from our surveys and pull-requests discussions (**RQ2**). Three researchers conducted together all the qualitative data. We, firstly, analyzed the answers from quasi-contributors. Afterwards, we analyzed the answers from integrators and pull-requests discussions to cross-validate and enrich the results.

The analysis of the open-ended questions of our quasi-contributors survey was conducted in three steps. In the first step, three researchers analyzed two sets of 20 answers, with the goal of better defining and discussing the codes applied. Each cycle was followed by a discussion, to reach consensus for the categorization of each item. In the second step, each researcher analyzed the rest of the answers independently, followed once again by a discussion, until reaching consensus for the categorization of each item. In the third step, the researchers analyzed the categories aiming to refine the classification and group related codes in more significant, higher level categories. In addition, we quantitatively analyzed closed-ended questions (**RQ3**) to understand developers' perceptions about nonacceptance.

To complement **RQ2**, we analyzed the discussions of pull-requests in our sample, and integrators answers to the survey. We made use of the previously identified categories to check whether the quasi-contributors perceptions would be confirmed. These two sets of data had also been analyzed by three researchers independently, followed by consensus discussions. In the results section, we highlight the main themes that emerged along with quotes extracted from the pull-requests and open questions from our survey. We chose quotes and cases based on their representativeness.

4 RESULTS

In this section, we report the results of our study grouped by each research question.

4.1 RQ1. How common are quasi-contributors and quasi-contributions?

We found a total of 10,099 unique quasi-contributors who have performed a total of 12,367 unmerged pull-requests. By comparison, the projects in our sample have a total of 14,623 actual contributors, who performed a total of 126,913 pull-requests. Figure 1 compares the number of quasi-contributors and actual contributors per project.

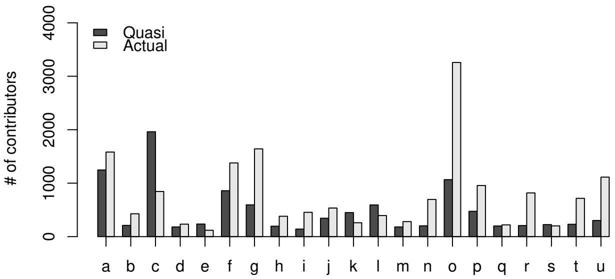

Figure 1: The number of quasi-contributors and actual contributors. From left to right, the projects are: a. `angular.js`, b. `bitcoin`, c. `bootstrap`, d. `caffe`, e. `d3`, f. `django`, g. `docker`, h. `flask`, i. `jenkins`, j. `joomla!`, k. `jquery`, l. `laravel`, m. `mongo`, n. `opencv`, o. `rails`, p. `react`, q. `redis`, r. `scikit-learn`, s. `spring-framework`, t. `tensorflow`, u. `kubernetes`.

As noted in Figure 1, three projects presented more than 1,000 quasi-contributors (`angular.js`, `bootstrap`, `rails`), whereas five projects have more than 1,000 actual contributors (`angular.js`, `django`, `docker`, `rails`, and `kubernetes`). Since 67% of projects hosted on GitHub have only 1 contributor (93% have 3 or less) [14], quasi-contributors could improve contributions and collaboration with GitHub projects.

In 5 out of the 21 analyzed projects, we found are more quasi-contributors than actual contributors. In some cases, the number of quasi-contributors is significantly higher: for instance, project `bootstrap` (c), which has 2.3× more quasi-contributors than actual ones (it has 1,962 quasi-contributors and 844 contributors). Project `d3` has 235 quasi-contributors and 119 actual contributors — 2.0× more quasi-contributors than actual ones. On average, there are 480.9 quasi-contributors per project (3rd quartile: 593.0, standard deviation: 459.0), and 730.4 actual contributors (3rd quartile: 844.0, standard deviation: 706.7).

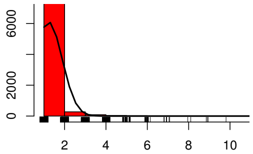

Figure 2: Quasi-contributors distribution

The identified quasi-contributors submitted 12,367 unmerged contributions. On average, a quasi-contributor tried 1.22 times (3rd quartile: 1.00, standard deviation: 0.65). The histogram (Fig. 2) presents the overall distribution of the pull-requests unmerged. This figure shows that 8,552 quasi-contributors performed a single attempt, *i.e.*, 84.68% of our dataset of quasi-contributors were at least interested in becoming casual-contributors [19]. Moreover, Figure 3 shows the distribution of the 15% of quasi-contributors that have performed two or more attempts. As this figure shows, project `joomla!` presented the greatest ratio of attempts per user: 1.52. That is, in this project, 95 quasi-contributors (27.61%) have performed two or more attempts. On the other hand, project `flask` had fewer attempts per quasi-contributors: only 6.6% tried more than once.

Figure 4 shows the distribution of quasi-contributions per quasi-contributor per analyzed project. As we can see in this figure, the

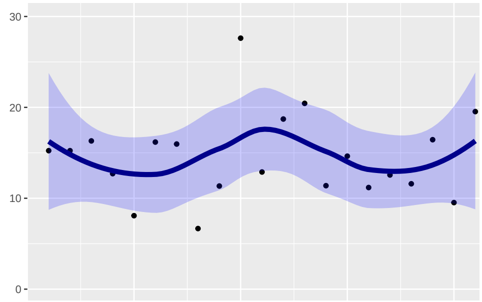

Figure 3: Percentage of quasi-contributors who have performed two or more attempts to contribute. Each point represents one analyzed project, following the same order of Figure 1.

average of quasi-contributions per project does not differ significantly (project `joomla!` (j) is the only one with an average higher than 1.5), except for the few outliers found. For instance, project `bootstrap` had a total of 2,430 unmerged pull-requests, proposed by 1,962 quasi-contributors. In this particular project, one quasi-contributor proposed 13 unaccepted pull-requests — the maximum number of pull-requests proposed by a single person in our dataset. On the other hand, project `flask` (h) unaccepted a total 214 pull-requests proposed by 195 quasi-contributors. When taking into account absolute numbers, project `bootstrap` (c) is the one with the greatest number of quasi-contributions and quasi-contributors (2,430 and 1,962). Other representative examples are the projects `angular` (a) (1,524 and 1,247, respectively) and `rails` (o) (1,289 and 1,066, respectively).

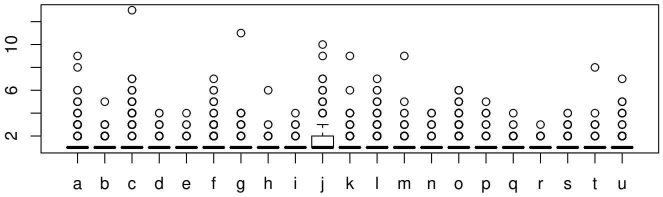

Figure 4: The distribution of quasi-contributions per quasi-contributor. Each boxplot represents one analyzed project, following the same order of Figure 1.

To complement this overview of quasi-contributors and quasi-contributions, for each analyzed project, we statistically compared some metrics (number of comments, review comments, commits, changed files, and line of codes added and deleted) from the unmerged and accepted pull-requests. In Table 2, we show the effect size of this comparison. Effect sizes are additionally colored on a gradient from blue to orange. Cells in blue highlight when the effect size is higher for the unmerged pull-request, whereas cells in orange otherwise. Additionally, green cells represent statistically significant differences.

Table 2: Effect size comparison between unmerged and merged pull-requests. EF means effect size and PV means p-value.

Projects	# Comments		# Files		# LoC Add		# LoC Del	
	ES	PV	ES	PV	ES	PV	ES	PV
`angular`	-0.42	0.00	0.49	0.00	0.48	0.00	0.49	0.00
`bitcoin`	0.11	0.00	0.26	0.00	0.14	0.00	0.26	0.00
`bootstrap`	-0.41	0.00	0.35	0.00	0.29	0.00	0.34	0.00
`caffe`	0.16	0.00	0.01	0.72	-0.01	0.39	0.13	0.12
`d3`	-0.35	0.00	0.01	0.38	0.01	0.39	0.01	0.12
`django`	-0.38	0.00	0.26	0.00	0.13	0.00	0.35	0.00
`docker`	0.00	0.87	0.22	0.00	0.19	0.00	0.31	0.00
`flask`	-0.38	0.00	0.00	0.98	-0.08	0.06	0.00	0.00
`jenkins`	0.00	0.00	0.26	0.00	0.21	0.00	0.32	0.00
`joomla!`	0.03	0.21	0.38	0.00	0.23	0.00	0.39	0.00
`jquery`	-0.38	0.00	0.53	0.00	0.50	0.00	0.56	0.00
`kubernetes`	0.00	0.00	0.22	0.00	0.22	0.00	0.33	0.00
`laravel`	-0.36	0.00	0.02	0.35	0.00	0.91	0.08	0.00
`mongo`	-0.77	0.00	0.03	0.54	-0.00	0.96	0.06	0.36
`opencv`	-0.24	0.00	0.18	0.00	0.17	0.00	0.31	0.00
`rails`	-0.28	0.00	0.13	0.00	0.04	0.05	0.27	0.00
`react`	-0.30	0.00	0.39	0.00	0.32	0.00	0.42	0.00
`redis`	0.20	0.00	0.00	0.92	0.00	0.93	0.09	0.04
`scikit-learn`	-0.07	0.08	0.12	0.00	-0.07	0.09	0.13	0.00
`spring`	-0.20	0.00	0.29	0.00	0.25	0.00	0.36	0.00
`tensorflow`	-0.26	0.00	0.28	0.00	0.18	0.00	0.30	0.00

From this table, we removed the "Review Comments" and "Commits" metrics because their effect sizes were negligible, in the majority of the cases. For instance, using the "review comments" metric, in only six projects (`angular.js`, `bitcoin`, `django`, `jenkins`, `react`, and `kubernetes`), we found a small effect size. The median value of "review comments" was zero for both groups. It is important to note that "comments" and "review comments" differ in the sense that the former are inside the pull-request (*e.g.,* to discuss why the pull-request is important), and the latter are inside the source code (*e.g.,* to suggest implementation changes).

Moreover, we note that, in 13 projects the "number of comments" was higher in unmerged pull-requests than in merged. Specifically, in 7 projects (`angular.js`, `bootstrap`, `d3`, `django`, `flask`, `jquery`, and `laravel`), we found a medium effect size, and one large effect size (`mongodb`). This finding is in line with recent literature that suggests that highly discussed pull-requests were much less likely to be accepted [31].

Finally, when observing the remaining three metrics ("changed files", 'lines of code added", and "lines of code deleted"), we found that, for most of the cases, merged pull-requests had higher values than unmerged ones. This result suggests that unmerged pull-requests tend to be smaller. For instance, merged pull-requests had a median value of 7,5 lines of codes added and 5 lines of code deleted (unmerged pull-requests had a median value of 3 and 1, respectively).

RQ1 Summary: Quasi-contributors are rather common. In our sample we found 10,099 quasi-contributors, and 14,623 actual contributors. Five projects have more quasi-contributors than actual ones. Most of the quasi-contributors (85%) try just once. Quasi-contributions are more commented and smaller than accepted contributions.

4.2 RQ2. Why were the quasi-contributions not accepted?

To understand the reasons why the pull-requests were not accepted, we qualitatively analyzed the answers to an open question (Q5) from our first survey. After analyzing and discussing the results, we found 19 different reasons mentioned by the quasi-contributors. The resulting reasons, with the number of mentions for each, are presented in Table 3. To analyze how the perceived reason affects the developers' degree of agreement, we also show the number of mentions for a given reason for those who disagreed with the pull-request's nonacceptance (in the third column). In the last column, we present the percentage of mentions of each category given by developers who disagreed.

From the table, it is possible to notice that the most common reason for nonacceptance from quasi-contributors perspective was that their pull-requests were **superseded/duplicated**. In general, the respondents answered that there was already something in place (*"Other pull-requests fixed the same issues as my pull-requests"*), or that someone else made a similar pull-request, which was accepted. An example was brought by a respondent who said *"The fix I submitted remained unmerged until someone else submitted the exact same fix … the integrators accepted their (identical) fix and closed mine."* Although 52 respondents mentioned this reason, only 7 of them (13.4%) disagreed with the nonacceptance.

Mismatch between developer's and team's vision/opinion was the second most mentioned reason for nonacceptance (45 mentions). In this case, a high number of quasi-contributors (27 of them, or 60%) disagreed with the decision. This reason includes the cases in which quasi-contributors thought something would be good, useful, or needed to be changed, while the integrators disagreed. There were cases in which the author of the pull-request agreed with the project members' positions, as reported by one respondent *"when you add a new feature to the project, your vision can be out of tune with the vision of the project's team, and this is natural"*. However, in 27 cases, this tag was associated with an answer from developers who disagreed with the nonacceptance, for example, one of them mentioned that *"The project decided that was not a bug they wanted to provide a fix for."*, and another, a little more nonconformist, answered that *"Because one person was skeptical."*.

We could also observe that **lack of interest from integrators** was presented as the reason for nonacceptance in 25 cases. In 18 (72%) cases, the developers did not agree with the way the decision about the pull-request was made. Many of them (13) had been simply ignored, and did not receive answers or reviews (e.g. *"It was ignored maybe because it was a very minor fix,"* and *"I did not receive answers"*), and some of them became upset, like the respondent who stated: *"That specific maintainer often ignores people's pull-requests and closes them without justification."* In other cases, quasi-contributors perceived that the community was not supportive enough to help them get their pull-request accepted, as reported by a respondent: *"The maintainer was unable to reproduce the bug...a greater effort to reproduce the issue could have been made."*

It was interesting to find **bureaucracy** as a perceived reason for nonacceptance (with a high disagreement with the nonacceptance: 83.3%). Regarding bureaucracy, one respondent mentioned, *"The process is too onerous, and bureaucratic"*. Quasi-contributors mentioned that some required steps in the process hindered their pull-request acceptance, for example, special types of sign-off, as mentioned by two developers: *"Introducing new features is problematic because the whole team needs to accept it"*. In this case, the developer mentioned that the pull-requests matched project requirements, passed tests, and still were not merged, because of these sign-off specificities. Another interesting finding relates to **license issues**. Six quasi-contributors mentioned signing an agreement or the impossibility of providing the proprietary code to reproduce the bug as the reason to nonacceptance.

Quasi-contributors also mentioned some better-received reasons, just as "superseded/duplicated pull-request". **Work in progress** as a reason was mentioned by seven developers, and none of them disagreed with the nonacceptance. In these cases, the quasi-contributors reported that something was part of a bigger change already under development, as explained by a respondent: *"There was major work in progress that conflicted with the pull-request and would finally also solve the problem."* Another reason with a relatively low disagreement with the nonacceptance was **PR not needed/not relevant**. Quasi-contributors that reported this reason appeared to be convinced that the proposed changed *"wasn't important enough to warrant merging"* or *"was an unnecessary change"*.

We also found quasi-contributors offering a *mea culpa* as the reason for nonacceptance; 20 respondents mentioned that the reason was **quasi-contributor's lack of experience/commitment**, like in this case: *"Because it was incomplete, and I never followed through on the feedback to complete it"*. **Not an optimal solution** was also a reason that (24) quasi-contributors assumed the pull-request was not good enough. One of them reported that the *"[my] fix was iffy"*, and other just answered: *"I made a mistake."*

An interesting observation is that 13 people (3.9%) mentioned that they did not know the exact reason (*e.g.*, *"I do not know, it's been a very long time since this pull-request."*). More interesting is observing that 9 of them did not agree with the nonacceptance. A possible explanation for those who do not know is that they, in fact, could not understand or did not accept the decision.

We also asked integrators their perception about nonacceptance. According to them, the most common reason for nonacceptance is **PR not needed/not relevant** (10 occurrences), for instance, "*[such contribution] solves the immediate/local problem but does not address the deeper systemic issue*". The second most common reason is **Guidelines not followed** (9 occurrences). According to one integrator, such guidelines can range from "*coding style, lack of tests, or messy versioning history*". Interestingly, none of the integrators assumed the *mea culpa*, *i.e.,* the **lack of interest from integrators** aforementioned, although one integrator raised the fact that there are very few integrators available for reviewing code: "*people love to contribute code, but it's harder to get people to spend time reviewing other people's code*". Instead, seven integrators also perceived this fact, as one respondent mentioned "*some [quasi-]contributors are not experienced with git, GitHub and/or C++*". According to one integrator, such lack of experience might explain the high number of **superseded/duplicated pull-request**, since quasi-contributors "*didn't look for existing patches*". Finally, two integrators consider small fixes, such as style or fixing typos, as **noisy**, as one respondent stated: "*Trivial PRs that are more trouble than they are worth (e.g., a non-native speaker attempting to add or improve comments)*".

Table 3: Self-perceived reason why pull-request was not merged

Reason for nonacceptance	mentions (# devs)	disagreed w/ nonacceptance	% of disagreement
Superseded/duplicated pull request	52	7	13.46%
Mismatch between developer's and team's vision/opinion	45	27	60.00%
PR not needed/not relevant	37	8	21.62%
Lack of interest from integrators	25	18	72.00%
Not an optimal solution	24	2	8.33%
Lack of experience/commitment from quasi-contributors'	20	4	20.00%
Did not answer	14	3	21.43%
Don't know	13	9	69.23%
Introduce side effects	12	2	16.67%
Lack of tests	11	2	18.18%
Incomplete change	10	4	40.00%
Work in progress	7	0	0.00%
Code/PR was obsolete	6	2	33.33%
Guidelines not followed	6	0	0.00%
License issues	6	3	50.00%
Not a bug	4	1	25.00%
Bureaucracy	6	5	83.33%
The project is too hard to contribute	3	1	33.33%
Lack of communication skills	2	1	50.00%

Ultimately, we manually investigated a representative sample of 263 pull-requests to cross-validate the results from our surveys about why the contributions were not accepted. We employed the same codes used in the surveys to analyze the quasi-contributions. In this analysis, we found that the most common reason for nonacceptance is **Superseded/Duplicated solution**[3] (32 occurrences), followed by **Quasi-contributors' lack of experience/commitment**[4] (25 occurrences) and **PR not needed/not relevant**[5] (22 occurrences). This finding is in sharp agreement with our surveys.

RQ2 Summary: We identified 19 reasons for nonacceptance. There is also a lack of communication, commitment, and experience. Integrators agree with quasi-contributors that not needed pull-requests are among the most common cause for nonacceptance. Manual investigation corroborated with the reasons for nonacceptance found at the surveys.

4.3 RQ3. How do quasi-contributors perceive nonacceptance?

To better understand quasi-contributors' perceptions about their unmerged pull-requests, we asked if they agreed with the integrators decision not to merge the pull-request (Q6), if this fact demotivated them to further contribute (Q7), and if the comments received were constructive (Q8). In addition, there were two open questions asking the developers about their motivation to contribute (Q4) and the why they think their pull-request was not merged (Q5).

In Figure 5, we provide the results for the "yes/no" based questions. Most of the developers agreed with the decision of having the pull-request unmerged (67.4%) and that the comments were constructive (88.8%). We also can notice that, for 69.7% of the respondents, having unmerged commits did not demotivate them from providing more pull-requests. However, the number of unmerged pull-requests, the 30.3% of the developers that reported demotivation, and the 32.6% that reported disagreement with the community decision are not negligible.

Table 4: Demotivation regarding nonacceptance

	Felt demotivated	
Agreed with the nonacceptance	**Yes**	**No**
Yes	37	178
No	**62**	43

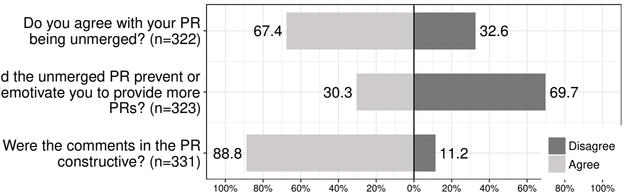

Figure 5: Perceptions about nonacceptance

Therefore, we took a closer look at these respondents to see how their answers are related. By analyzing Table 4, we observe that 99 respondents did not agree with the pull-request nonacceptance, and 62.6% of them (62) answered that this fact demotivated or prevented them from placing another pull-request. This can indicate that not accepting a pull-request can be a driving force in demotivating newcomers, confirming what was stated by Karl Fogel: *"if a project doesn't make a good first impression, newcomers may wait a long time before giving it a second chance"* [6]. This finding might also explain why most of the quasi-contributors placed just a single quasi-contribution. Interestingly, some integrators are aware of such side effect. Four respondents mentioned that deal with personalities and opinions is one of the main challenges that they face, as one of them highlighted: *"We want to encourage contributions and refusing one, even for very good reasons, can be discouraging to new contributors"*.

RQ3 Summary: 32% of the quasi-contributors do not agree with the decision of not having their contribution accepted — for 19% of quasi-contributors that did not concur also felt demotivated or prevented to place additional contributions. 12% of the quasi-contributors reported that the feedback from the code review was not constructive.

5 DISCUSSION

In this section, we provide additional discussion on the data presented in the previous section.

Quasi-contributors should stand for their contributions. We found cases in which the quasi-contributor disappeared shortly after proposing the pull-request. Even if a proposed pull-request had a robust implementation, integrators were unable to accept since the proposed changes were not addressed during the code review process. We observed this by manually analyzing pull-requests,

[3] https://github.com/antirez/redis/pull/1160
[4] https://github.com/django/django/pull/6174
[5] https://github.com/moby/moby/pull/14576

and it was confirmed by the quasi-contributors who reported they did not make the requested change (category **quasi-contributor's lack of experience/commitment** in Table 5).

The eagerness to contribute is not always helpful. We noticed that some contributions were not accepted because the quasi-contributor failed to understand the contribution process adopted by the project (*e.g.,* contributions style) or internal implementation details (*e.g.,* do not know the difference between two variables[6]). If quasi-contributors placed additional care in understanding such details, their contributions might have higher chances of acceptance.

The fast development cycle hinders external contributions. External contributors face an additional challenge since contributions proposed to an outdated code are likely not accepted (category **Code/PR was obsolete**). Further complicating this matter is that outdating can happen in a matter of days, as we noticed in one review: "*Thx for the PR, but this change is not valid. In the latest tutorial version (updated a few days ago), the app is served from /app, which means that you don't need /app in the URL*".

Typos are sometimes hard to fix. We found cases in which quasi-contributions proposed simple typo fixes that were not accepted by integrators. As one extreme example, one quasi-contributor added one single comma to the documentation[7], but integrators found that it was better to keep the current wording than to accept additional contributions (category **Mismatch between developer's and team's vision/opinion**). Yet, in our survey, two integrators mentioned that such small contributions are **noisy**.

Some PRs were not accepted but were implemented. We found cases where the proposed pull-request was not accepted by the integrators, although integrators got inspired and implemented the proposed solution themselves[8]. This was also mentioned by 13 quasi-contributors who answered our survey. It is important to highlight that 10 out of these 13 disagreed with nonacceptance.

No tests, no fun. In some projects (*e.g.,* bootstrap), if the proposed contribution does not come with tests, the pull-request is immediately closed – no questions asked (category **Lack of tests**). Although newcomers are encouraged to file additional pull-requests containing tests, we observed that few of them do so.

Integrators are not always kind. Although integrators mentioned in our survey that *"we try to be as kind as possible"*, we found examples of impolite tone while dealing with quasi-contributors (*e.g., "Not crazy about adding an option for this."*). The category **Lack of interest from integrators** was also recurrent in our survey with quasi-contributors and when analyzing the repositories. Impolite tone and lack of interest may demotivate quasi-contributors.

Truck Factor does not consider the number of developers willing to contribute. The Truck Factor (TF) calculates "*the number of people on your team that have to be hit by a truck (or quit) before the project is in serious trouble*" [34]. When empirically evaluated, some authors observed that some highly popular projects have indeed a really small truck factor, which places a strong dependency on specific contributors [1]. For instance, the project d3 has TF of 1, which means that a single contributor is responsible for managing most of the tasks related to the project. Notwithstanding, d3 has 235 quasi-contributors. That is, this low TF is not related to the number of external contributors willing to contribute to the project.

"Scratching itches" plays a role. In our first survey, we asked quasi-contributors what motivated them to contribute to the OSS project. The four most common motivations are: (i) Fixing a bug (102 occurrences), (ii) Scratching own itch (71 occurrences), (iii) Improving the project (52 occurrences), and (iv) Altruism/Giving back (51 occurrences). This result shows that a great part of the respondents was attempting to **fix a bug** and **scratch their own itch**. We observed that the results align with those presented by Pinto *et al.* [19], who analyzed the motivation behind casual contributions.

6 IMPLICATIONS

This research has implications for different stakeholders.

OSS integrators: Since both integrators and quasi-contributors agree that pull-requests that do not follow the guidelines are more likely to be rejected (*e.g.,* "*Our project has strict rules for contributions and PRs that do not follow the rules are not merged*"), integrators can clearly state what are their project norms upfront. We also found that nonacceptance might demotivate quasi-contributors, incurring in fewer contributions (**RQ3**). To prevent such behavior, OSS project members should (1) state and follow a code of conduct and (2) be kind and respectful. Finally, a clear roadmap describing the future plans of the project might avoid **PR not needed/relevant**.

Newcomers to OSS project: Our surveys and our manual analysis showed evidence regarding a lack of commitment of quasi-contributors (**RQ1–RQ2**). Therefore, it is important for newcomers to work closely with integrators, and, when necessary, stand and argue for their contributions. Newcomers should also take a moment to investigate whether there are existing pull-requests proposing the same contribution, and, thus, avoid duplication. Newcomers not only should back up their contributions with tests but also make sure that their contributions do not **introduce side effects**.

CS educators: Some OSS projects do not even review contributions that do not come with tests (Section 5). Still, some tests are hard to implement, as one integrator stated: "*Typical case is contributor figure out a way to implement the change but not how to implement the tests.*" Educators can these scenarios of real-world tests along with students. Additionally, software engineering educators can bring non-accepted pull-requests, those considered "not an optimal solution" (**RQ2**), to the classroom to engage students to discuss and, eventually, propose better solutions. Finally, since integrators and quasi-contributors reported a lack of experience with git and GitHub tools, it is important for the educators to introduce such tools in introductory programming courses.

Researchers: We found 25 quasi-contributors pointing a lack of interest from integrators. Still, two integrators also pointed out a lack of integrators available to review. Researchers can take advantage of this fact and introduce mechanisms to automate/simplify code review, for example, verifying if the proposed pull-request is touching an outdated code-base or generating test data based on diff files. Researchers can also create techniques to compare pull-requests and warn quasi-contributors if similar pull-requests are found. Researchers can also introduce tools that identify impolite tone and suggest alternatives.

[6] https://github.com/angular/angular.js/pull/10810
[7] https://github.com/angular/angular.js/pull/9168
[8] https://github.com/twbs/bootstrap/pull/1900

7 THREATS TO VALIDITY

In an empirical study, there are always limitations. We discuss our limitations in term of internal and external threats to validity.

7.1 Internal Validity

First, one might argue that we analyzed too few projects, therefore, limiting the generalization of our results. However, the selected OSS projects are diverse in terms of *domain*, `popularity`, and `activity`. When quantitatively analyzing data from these project, we used statistical methods to mitigate the threats of generalizing data based on our personal hypothesis. Second, since we leveraged qualitative research methods to categorize the open-ended answers to our surveys, as well as unmerged pull-requests, we may have introduced categorization bias. To mitigate this bias, we conducted this process in pairs and carefully discussed categorization among the authors. Still regarding our surveys, the order that the questions presented may have influenced the way they answered. Instead of randomizing the questions, we tried to order the questions based on the natural sequence of actions to help respondents understand the questions' context. Moreover, we made our survey as short as possible, none of the questions were mandatory, the responses were anonymous, and participation was voluntary. We also employed well-known survey principles [15, 24].

7.2 External Validity

By conducting this study, it was possible to observe that `git` and GitHub data can be mismatching, which can lead to misinterpretation. We noticed that, even with the facilities introduced by the pull-based model, some projects still do not use this approach, preferring to merge the contributions via git Command Line Interface (CLI). Even using the CLI, this can be done in several ways, which can lead to losing trace of a contribution, as reported by a survey respondent: *"eventually they copy the related code and inject it into the project by their own developers"*. In our study, we identified false-positives in two ways. First, we compared whether the commits hash that appears at the pull-request exist in the git repository. Also, we queried GitHub users' API to double-check whether the quasi-contributor authored any commit in a given repository. This API lists commits that were authored by a contributor under different SHA identifiers. We removed any duplication. Second, our survey asked whether we correctly identified the developer as a quasi-contributor. Some respondents (13%) said that we did not. We asked the respondents to help us by telling why they believe this happened, and a summary of their answers is presented in Table 5. We removed these cases from the analysis and implemented heuristics to catch similar cases when possible.

Additionally, our results only apply to developers who attempted to contribute to OSS projects hosted on GitHub. They do not cover software developers in other source code hosting websites. Our results are limited by our selection of OSS projects, which one might argue is small or non-representative. However, we argue that the selected projects are diverse in several dimensions (Section 3.2 provide greater details). Moreover, quasi-contributors are particularly relevant to OSS projects; in proprietary projects, although developers may have similar challenges to contribute, they are required to contribute. This distinction made proprietary projects unsuitable for this study. Moreover, we likely did not discover all possible characteristics of quasi-contributions. With our methodology and infrastructure [9], we expect similar analysis to be conducted in the future when such characteristics become relevant. We also expect to understand the integrators' and the communities' perspectives about the quasi-contributions. Still, comparing the quasi-contributors' unmerged pull-requests to the actual-contributors' unmerged pull-requests can shed additional light to the phenomenon.

Table 5: Participants perception on why mistakenly identified their pull-requests as Unmerged

Reason	# answers
No Answer/ Do not know/ Did not give a reason	19
Amended in another commit/squashed	11
Manual merge/ Done outside GitHub workflow	4
Self-closed	2
PR done during migration and merged outside GitHub	1
PR went to another repo/branch	2

8 CONCLUSION

For OSS projects remain sustainable and evolve, it is important that new contributors onboard the project fixing bugs and proposing new features. However, the path to becoming an OSS contributor is not always flowery. In this paper, we investigate quasi-contributors, that is, OSS contributors that tried to contribute, but did not succeed. Through quantitative and qualitative analysis from software repositories and two surveys with quasi-contributors and integrators, we found that quasi-contributors are rather common, although the majority of them only tried once. The most common reason for nonacceptance was "mismatch between developer's and team's vision/opinion", followed by reasons related to a relationship with the community, while others attributed fault to the developers (either quasi-contributors or integrators). Still, about one-third of the developers disagreed with their nonacceptance and declared the nonacceptance demotivated or prevented them from placing another pull-request. Our results can be relevant for developers interested in contributing to OSS projects, by bringing the common issues that can lead to pull-request nonacceptance. In addition, the outcomes can benefit those interested in building sustainable communities and fostering contributions, since our results include reasons that can potentially scare external members away.

For future work, we plan to propose and evaluate a virtual assistant which will be in charge of helping newcomers to join a community. Such virtual assistant, while offering guidance for newcomers to overcome their first barriers, would lower the effort required on the integrators side.

ACKNOWLEDGMENTS

We thanks the reviewers for their valuable comments. This work is supported by the CNPq (Grants # 406308/2016-0 and # 430642/2016-4); PROPESP/UFPA; and FAPESP (Grant # 2015/24527-3).

[9] https://doi.org/10.5281/zenodo.1154906

REFERENCES

[1] Guilherme Avelino, Leonardo Teixeira Passos, André C. Hora, and Marco Tulio Valente. 2016. A novel approach for estimating Truck Factors. In *24th IEEE International Conference on Program Comprehension, ICPC 2016, Austin, TX, USA, May 16-17, 2016*. 1–10.

[2] H. Borges, A. Hora, and M. T. Valente. 2016. Understanding the Factors That Impact the Popularity of GitHub Repositories. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 334–344. https://doi.org/10.1109/ICSME.2016.31

[3] Jailton Coelho and Marco Tulio Valente. 2017. Why Modern Open Source Projects Fail. In *25th International Symposium on the Foundations of Software Engineering (FSE)*. 186–196.

[4] Luiz Felipe Dias, Igor Steinmacher, Gustavo Pinto, Daniel Alencar da Costa, and Marco Aurélio Gerosa. 2016. How Does the Shift to GitHub Impact Project Collaboration?. In *2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016, Raleigh, NC, USA, October 2-7, 2016*. 473–477.

[5] Susan Elliott Sim and Richard C. Holt. 1998. The Ramp-up Problem in Software Projects: A Case Study of How Software Immigrants Naturalize. In *20th International Conference on Software Engineering (ICSE '98)*. 361–370.

[6] Karl Fogel. 2013. *Producing Open Source Software: How to Run a Successful Free Software Project* (first ed.). O'Reilly Media. http://www.producingoss.com/ .Accessedon01-15-2015

[7] Georgios Gousios and Alberto Bacchelli. 2016. Work Practices and Challenges in Pull-based Development: The Contributor's Perspective. In *ICSE*. 358–368.

[8] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An Exploratory Study of the Pull-based Software Development Model. In *36th International Conference on Software Engineering (ICSE 2014)*. ACM, New York, NY, USA, 345–355. https://doi.org/10.1145/2568225.2568260

[9] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An Exploratory Study of the Pull-based Software Development Model. In *36th International Conference on Software Engineering (ICSE 2014)*. 345–355.

[10] Georgios Gousios, Andy Zaidman, Margaret-Anne D. Storey, and Arie van Deursen. 2015. Work Practices and Challenges in Pull-Based Development: The Integrator's Perspective. In *ICSE*. 358–368.

[11] Vincent J. Hellendoorn, Premkumar T. Devanbu, and Alberto Bacchelli. 2015. Will They Like This?: Evaluating Code Contributions with Language Models. In *12th Working Conference on Mining Software Repositories (MSR '15)*. IEEE Press, Piscataway, NJ, USA, 157–167. http://dl.acm.org/citation.cfm?id=2820518.2820539

[12] Guido Hertel, Sven Niedner, and Stefanie Herrmann. 2003. Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel. *Research Policy* 32, 7 (2003), 1159 – 1177. Open Source Software Development.

[13] Yujuan Jiang, Bram Adams, and Daniel M. Germán. 2013. Will my patch make it? and how fast?: case study on the Linux kernel. In *10th Working Conference on Mining Software Repositories, MSR '13, San Francisco, CA, USA, May 18-19, 2013*. 101–110.

[14] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. 2016. An in-depth study of the promises and perils of mining GitHub. *Empirical Software Engineering* 21, 5 (2016), 2035–2071. https://doi.org/10.1007/s10664-015-9393-5

[15] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K. El Emam, and J. Rosenberg. 2002. Preliminary guidelines for empirical research in software engineering. *Software Engineering, IEEE Transactions on* 28, 8 (Aug 2002), 721–734. https://doi.org/10.1109/TSE.2002.1027796

[16] Nora McDonald and Sean Goggins. 2013. Performance and Participation in Open Source Software on GitHub. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems (CHI EA '13)*. ACM, New York, NY, USA, 139–144. https://doi.org/10.1145/2468356.2468382

[17] Meiyappan Nagappan, Thomas Zimmermann, and Christian Bird. 2013. Diversity in Software Engineering Research. In *2013 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2013)*. 466–476.

[18] Rohan Padhye, Senthil Mani, and Vibha Singhal Sinha. 2014. A Study of External Community Contribution to Open-source Projects on GitHub. In *11th Working Conference on Mining Software Repositories (MSR 2014)*. ACM, New York, NY, USA, 332–335. https://doi.org/10.1145/2597073.2597113

[19] G. Pinto, I. Steinmacher, and M. A. Gerosa. 2016. More Common Than You Think: An In-depth Study of Casual Contributors. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. 112–123. https://doi.org/10.1109/SANER.2016.68

[20] Mohammad Masudur Rahman, Chanchal K. Roy, and Jason A. Collins. 2016. CoRReCT: Code Reviewer Recommendation in GitHub Based on Cross-project and Technology Experience. In *38th International Conference on Software Engineering Companion (ICSE '16)*. ACM, New York, NY, USA, 222–231. https://doi.org/10.1145/2889160.2889244

[21] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. 2014. A Large Scale Study of Programming Languages and Code Quality in Github. In *22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*. 155–165.

[22] P. C. Rigby and M. A. Storey. 2011. Understanding broadcast based peer review on open source software projects. In *2011 33rd International Conference on Software Engineering (ICSE)*. 541–550. https://doi.org/10.1145/1985793.1985867

[23] J. Romano, J.D. Kromrey, J. Coraggio, and J. Skowronek. 2006. Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other surveys?. In *annual meeting of the Florida Association of Institutional Research*. 1–3.

[24] E. Smith, R. Loftin, E. Murphy-Hill, C. Bird, and T. Zimmermann. 2013. Improving developer participation rates in surveys. In *CHASE*. 89–92.

[25] Daricélio Moreira Soares, Manoel Limeira de Lima Júnior, Leonardo Murta, and Alexandre Plastino. 2015. Acceptance Factors of Pull Requests in Open-source Projects. In *30th Annual ACM Symposium on Applied Computing (SAC '15)*. ACM, New York, NY, USA, 1541–1546. https://doi.org/10.1145/2695664.2695856

[26] Igor Steinmacher, Ana Paula Chaves, Tayana Conte, and Marco AurÃĺlio Gerosa. 2014. Preliminary empirical identification of barriers faced by newcomers to Open Source Software projects.. In *28th Brazilian Symposium on Software Engineering (SBES '14)*. IEEE Computer Society, 1–10.

[27] Igor Steinmacher, Tayana Conte, and Marco AurÃĺlio Gerosa. 2015. Understanding and Supporting the Choice of an Appropriate Task to Start With In Open Source Software Communities. In *48th Hawaiian International Conference in Software Systems (HICSS '15)*. 1 – 10.

[28] Igor Steinmacher, Tayana Conte, Marco AurÃĺlio Gerosa, and David F. Redmiles. 2015. Social Barriers Faced by Newcomers Placing Their First Contribution in Open Source Software Projects. In *18th ACM Conference on Computer Supported Cooperative Work & Social Computing (CSCW '15)*. ACM, New York, NY, USA, 1–13.

[29] Anselm Strauss and Juliet M. Corbin. 2007. *Basics of Qualitative Research : Techniques and Procedures for Developing Grounded Theory* (3rd ed.). SAGE Publications.

[30] Y. Tao, D. Han, and S. Kim. 2014. Writing Acceptable Patches: An Empirical Study of Open Source Project Patches. In *2014 IEEE International Conference on Software Maintenance and Evolution*. 271–280. https://doi.org/10.1109/ICSME.2014.49

[31] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Influence of social and technical factors for evaluating contribution in GitHub. In *ICSE*. 356–366.

[32] Peter Weißgerber, Daniel Neu, and Stephan Diehl. 2008. Small patches get in!. In *2008 International Working Conference on Mining Software Repositories, MSR 2008 (Co-located with ICSE), Leipzig, Germany, May 10-11, 2008, Proceedings*. 67–76.

[33] D.S. Wilks. 2011. *Statistical Methods in the Atmospheric Sciences*. Academic Press. https://books.google.com.br/books?id=IJuCVtQ0ySIC

[34] Laurie Williams and Robert Kessler. 2002. *Pair Programming Illuminated*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[35] Yue Yu, Huaimin Wang, Vladimir Filkov, Premkumar Devanbu, and Bogdan Vasilescu. 2015. Wait for It: Determinants of Pull Request Evaluation Latency on GitHub. In *12th Working Conference on Mining Software Repositories (MSR '15)*. IEEE Press, Piscataway, NJ, USA, 367–371. http://dl.acm.org/citation.cfm?id=2820518.2820564

[36] Yue Yu, Huaimin Wang, Gang Yin, and Tao Wang. 2016. Reviewer Recommendation for Pull-requests in GitHub. *Inf. Softw. Technol.* 74, C (June 2016), 204–218. https://doi.org/10.1016/j.infsof.2016.01.004