# Opportunities and Challenges in Repeated Revisions to Pull-Requests: An Empirical Study

ZHIXING LI, National University of Defense Technology, China
YUE YU*, National University of Defense Technology, China
TAO WANG, National University of Defense Technology, China
SHANSHAN LI, National University of Defense Technology, China
HUAIMIN WANG, National University of Defense Technology, China

***Background:*** The Pull-Request (PR) model is a widespread approach adopted by open source software (OSS) projects to support collaborative software development. However, it is often challenging to continuously evaluate and revise PRs in several iterations of code reviews involving technical and social aspects. ***Aim:*** Our objective is twofold: identifying best practices for effective collaboration in continuous PR improvement and uncovering problems that deserve special attention to improve collaboration efficiency and productivity. **Method:** We conducted a mixed-methods empirical study of repeatedly revised PRs (*i.e.*, those that have undergone a high number of revisions). Historical trace data of five long-lived popular GitHub projects were used for manual investigation of practices for requesting changes to PRs and reasons for nonacceptance of repeatedly revised PRs. Surveys of OSS practitioners were conducted to evaluate the results of manual analysis and to provide additional insights into developers' willingness regarding PR revisions and factors causing avoidable revisions in practice. **Results:** The main results of our research were as follows: (1) We identified 15 code review practices for requesting changes to PRs, among which practices with respect to explaining the reasoning behind requested changes and tracking the progress of PR review and revision were undervalued by reviewers; (2) While submitters can in general undergo 1-5 rounds of revisions, they are willing to offer more revisions when they are in a friendly community and receive helpful feedback; (3) We revealed 11 factors causing avoidable revisions regarding to reviewers' feedback, code review policy, pre-submission issues, and implementation of new revisions; and (4) Nonacceptance of repeatedly revised PRs was due mainly to inactivity of submitters or reviewers and being superseded for better maintenance. Finally, based on these findings, we proposed recommendations and implications for OSS practitioners and tool designers to facilitate efficient collaboration in PR revisions.

CCS Concepts: • **Software and its engineering** → **Collaboration in software development**.

Additional Key Words and Phrases: Pull-Request, Code Review, Repeated Revision, Open Source Software

---

*Corresponding author: Yue Yu, yuyue@nudt.edu.cn.
Authors' addresses: Zhixing Li, lizhixing15@nudt.edu.cn, National University of Defense Technology, Changsha, China; Yue Yu, yuyue@nudt.edu.cn, National University of Defense Technology, Changsha, China; Tao Wang, taowang2005@ nudt.edu.cn, National University of Defense Technology, Changsha, China; Shanshan Li, shanshanli@nudt.edu.cn, National University of Defense Technology, Changsha, China; Huaimin Wang, hmwang@nudt.edu.cn, National University of Defense Technology, Changsha, China.

---

**111**

## 1 INTRODUCTION

Open source software (OSS) developers collaborate and coordinate with each other in an increasingly efficient way, relying on a wide range of mechanisms and tools provided by the community [30, 38, 97, 104, 108]. To contribute code to OSS projects, developers often use the Pull-Request (PR) model [38] in which developers first fork a repository, then make code changes locally, and finally submit a PR for integration of the changes. The PR model is associated with higher information centralization, process automation, and tool integration [38, 104], which encourage an increasing number of developers to submit PRs to OSS projects. In GitHub, the most popular online collaborative development site, more than 87 million PRs were merged in 2019 alone [1].

Despite the huge number of PRs, the process is most effective when every PR can be incorporated correctly, rigorously and quickly. However, PRs submitted from the community are rarely perfect and ready to merge at the first submission. Reviewers (generally referring to the project maintainers/core members) need to inspect and discuss proposed changes to ensure software quality [68]. And PR submitters are expected to react to the received feedback and carry out the requested changes. In practice, it takes a series of iterations for PR submitters and reviewers to collaboratively revise a PR until it can be merged into the code base [86]. Sometimes, the number of revision iterations can be high, that is, a PR is repeatedly revised. For example, a PR [2] of the Elasticsearch project was revised 36 times.

Multiple iterations of PR revisions can result in benefits. For example, reviewers can thoroughly identify potential defects and quality problems in PRs [67], which reduces the possibility of mistakenly merging problematic PRs [51]. In the meantime, as code review is a knowledge transfer process [4], PR submitters have a chance to learn more about the project and acquire programming skills and knowledge, as said by participants in our survey: *"It's the nature of open source contribution and an opportunity to learn together"* and *"Repeated rounds generally help to avoid them in the future"*. More importantly, repeated revisions to PRs highlight reviewers' success in promoting the continuous improvement of PRs, given that PR evaluation is a complex process involving technical and social aspects [50, 101, 113], and many PRs are abandoned by their submitters [62].

However, repeated revisions come at a cost. The time consumed in waiting for the next iteration may delay the integration of a PR. The workload of reviewers is increased because they need to maintain the context between revisions [56, 103]. Additionally, the consumption of continuous integration (CI) [104] resource in repeated revisions is non-negligible since each new revision to a PR automatically triggers CI build and testing jobs. Furthermore, repeatedly asking for changes in an overly long review process can cause frustration and potentially discourage some contributors, especially newcomers and casual contributors [60], and scare them away.

Although a number of studies have been performed on code review and PR evaluation [4, 6, 12, 13, 56, 67, 68, 76, 101, 116], very few have examined repeated revisions to PRs. Considering the challenges in conducting successful reviews [40, 56, 62], the lengthy back-and-forth interactions around repeatedly revised PRs can reveal important social and technical dynamics as they facilitated multiple revisions in an intensive review process. By systematically studying repeated revisions to PRs, we can achieve a more comprehensive understanding of the collaboration between PR submitters and reviewers. Moreover, repeated revisions are more likely to indicate a higher complexity of the code. An investigation of this phenomenon can provide valuable insights into how to properly and efficiently review complex PRs in an iterative process. Additionally, it also helps to uncover needs for tools and mechanisms that can improve the effectiveness and efficiency of PR revisions. To this end, we conducted an in-depth empirical study by adopting a mixed-methods approach. We

---

[1]https://octoverse.github.com/2019/
[2]https://github.com/elastic/elasticsearch/pull/15125

collected a set of repeatedly revised PRs from 5 popular OSS projects hosted on GitHub. Based on historical trace data on PR reviews, we manually analyzed the practices for requesting changes to PRs (**RQ1**) and the reasons for nonacceptance of repeatedly revised PRs (**RQ4**). We also surveyed OSS practitioners to evaluate the results of manual inspection and to gain additional insights into submitters' willingness regarding PR revisions (**RQ2**) and avoidable revisions in practice (**RQ3**).

From our analysis, we make the following contributions: (1) we identify a wide range of code review best practices for requesting changes to PRs that OSS reviewers can follow to facilitate the continuous improvement of PRs; (2) we reveal situations in which submitters are more willing to perform the requested changes, which has implications for reviewers in terms of how to better manipulate and lead code review processes; (3) we uncover factors that cause avoidable revisions, which may inspire practitioners and tool builders to reduce preventable revisions; and (4) we revisit reasons for PR nonacceptance in the context of repeated revisions, which increases practitioners' and researchers' awareness towards the inefficiencies in current OSS collaboration.

## 2 BACKGROUND AND RELATED WORK

This section describes the background on the PR workflow and presents related work.

### 2.1 Background: GitHub PR workflow

Figure 1 illustrates the entire life cycle and workflow of a PR in GitHub. If developers want to contribute code to a GitHub project, they need to first *fork* (*i.e.*, clone) the project repository. Next, developers *edit* the forked repository and make code changes locally. After finishing and committing local work, developers *submit* a PR for the integration of proposed changes. Each new PR as well as the subsequent commits pushed to the PR automatically triggers the CI service to run. In the meantime, project team members and community developers manually *review* the code changes in the PR and *comment* on how to improve it. After receiving reviewers' change requests, PR submitters *revise* the code accordingly and re-push the revised code to *update* the PR. However, popular projects can have many developers submitting PRs in a given period of time. If the main branch of the original repository has merged contributions from other developers during PR review, PR submitters generally need to *sync* the latest code to their local branch before updating the PR. The updated PR would then be examined by reviewers once again. This iterative review process is repeated until the reviewers consider that the PR is good and can be *accepted* or should be *rejected*.

### 2.2 Related work

We first present previous research on code review in OSS community. Additionally, we consider our work from the broad perspective of feedback-based revision, coalescing relevant literature from other feedback contexts.

#### 2.2.1 Code review in OSS community

We present four aspects of previous research on the area of code review and PR evaluation corresponding to our research topics.

***Effectiveness of code review.*** Submitters improve their PRs in response to reviewers' comments until the changes are approved. Therefore, reviewers' responses are critical to PR improvement. Bosu *et al.* [13] interviewed Microsoft developers to investigate what factors make a review comment useful to developers. They reported that comments were considered as useful if they pointed out issues related to functional defects and validations or if they helped developers to learn the project. Similarly, Kononenko *et al.* [56] conducted a survey with OSS developers to understand their perception of code review quality and found that a well-done code review was expected to
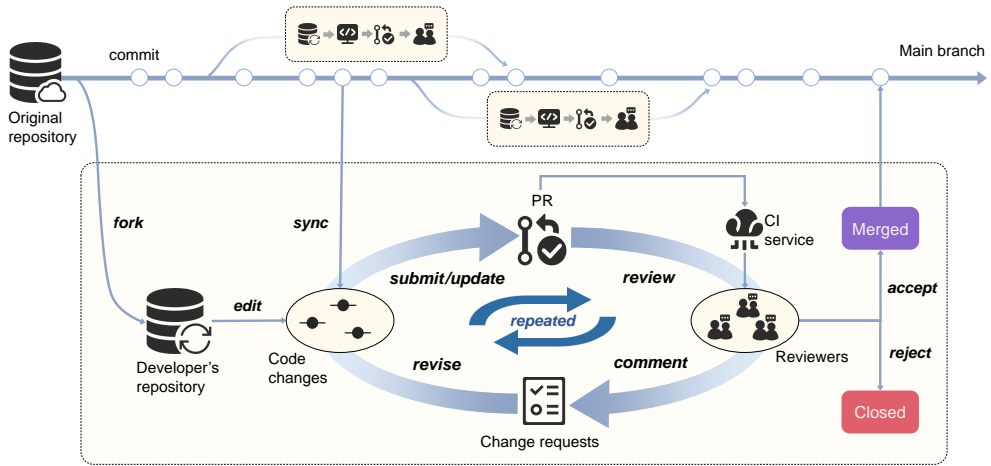
Fig. 1. The entire life cycle and workflow of a PR in GitHub.

be clear and thorough. Rahman *et al.* [82] compared useful and not useful review comments and observed significant differences in terms of code element ratio, stop word ratio, conceptual similarity, and developers' prior experience with the changed files. Recently, Efstathiou *et al.* [27] proposed leveraging linguistic information to assess the usefulness of review comments. Researchers also designed approaches to automatically predict the usefulness of code review comments [13, 82].

Previous studies have focused mainly on the content of review comments and their usefulness; however, few have investigated reviewers' practices in a systematic way. Although Macleod *et al.* [63] compiled a set of code review best practices, their study was based on development teams at Microsoft and they did not pay sufficient attention to the behavior of requesting changes particularly. Repeatedly revised PRs, in which reviewers have successfully requested changes in a high number of iterations, might provide valuable insights and useful guidelines for effectively requesting revisions to PRs.

***Outcome of code review.*** In the literature, considerable attention has been paid to the outcome of code review, *i.e.*, what problems reviewers have found in code changes. Mäntylä *et al.* [64] classified review comments from industrial developers and students and built a detailed taxonomy of defects types consisting of functional and evolvability defects. Their quantitative exploration showed that evolvability defects are far more popular than functional defects, with a 75:25 ratio. Subsequent studies based on Microsoft projects [4] and OSS projects [6, 76] complemented the classification proposed by Mäntylä *et al.* [64] and confirmed their quantitative findings. Additional research has invested specific defects in depth. For example, Zou *et al.* [118] manually analyzed Java code style documents defined by two famous technical companies and extracted 37 code style criteria. Zampetti *et al.* [114] investigated PR discussions related to CI testing and defined a taxonomy of build failures. Furthermore, some researchers studied on how to reduce reviewers' effort by automatically fixing identified defects [76, 92].

Although the above studies analyzed the types of changes in depth from a technical perspective, and some proposed automation solutions to accommodate certain types of changes, little research has been conducted from a practical perspective to investigate whether some revisions are unnecessary and could have been avoided in practice.

***Continuous participation of developers.*** The sustainability of many community-based OSS projects replies on the continuous participation of voluntary developers. The motivations driving developers to participate in OSS development and the factors affecting their engagement with projects have been widely studied in previous work. According to early research on this topic [58, 111], OSS developers were commonly motivated by learning, enjoyment, community identification, user need, career advancement, *etc.*. For casual contributors in particular, their participation was usually motivated by personal needs [60, 80], *e.g.*, fixing a bug that directly affected them. With the increasing involvement of companies in OSS communities, more developers were paid to contribute [85]. However, developers can face a variety of challenges, some of which might lead them to disengage from the projects. Regarding personal reasons, lack of time [48, 60, 62, 69, 80], lack of interest [48, 62, 69], and limited knowledge [60, 62, 80] were among the main reasons hindering developers from active and continuing participation. In terms of project-related factors, participation process [62, 80], development activeness [60, 69], and popularity [81] were found to have an influence on developers' engagement with projects. Regarding social issues, lack of response [48, 60, 62], lack of support [62, 69, 96], bad attitude [60], and disagreement [62] may prevent developers from contributing more.

While prior research has extensively studied developers' engagement in OSS communities, we still know relatively little about developers' patience through the continuous improvement of a single task and what factors can affect it.

***Nonacceptance of PRs.*** Not all PRs submitted from the community can be merged for various reasons. Gousios *et al.* [38] manually classified hundreds of PRs to explore the reasons for closing a PR. They identified a total of 9 nonacceptance reasons, and the dominant ones were related to the distributed nature of PR development model (*e.g.*, duplicate and conflict). Researchers have also explored developers' perceptions of the reasons for PR nonacceptance. Pham *et al.* [79] interviewed GitHub project owners and reported that project owners examined many factors when evaluating PRs, such as the trustworthiness of contributors and the size, type, and target of changes. Gousios *et al.* [40] conducted a survey of GitHub reviewers and found that quality, degree of fit to the project roadmap and technical design were the most common factors affecting reviewers' decisions. Steinmacher *et al.* [95] performed a survey of contributors and observed that the most frequently mentioned reason for PR nonacceptance was that PRs were superseded. In addition to qualitative analyses, extensive quantitative research has been conducted on the association between PR acceptance and various metrics, including project characteristics [38, 101], developer characteristics [50, 99], patch characteristics [38, 101, 118], social factors [101], and review activities [38, 101, 113].

Our paper differs from previous research in that we are particularly interested in understanding the nonacceptance of repeatedly revised PRs rather than general PRs. For repeatedly revised but unmerged PRs, their submitters and reviewers have invested considerable time and effort in multiple rounds of reviews and revisions; therefore the reasons for the nonacceptance of such PRs are worthy of investigation.

### 2.2.2 Feedback in other contexts

In addition to OSS contribution, many contexts also use feedback as an effective quality assurance method, including scientific publishing, student learning, and software maintenance.

***Review of scientific papers.*** In the scientific publishing process, peer review is an established method of assisting the board of editors to make decisions and helping the authors to revise their papers [22, 106]. Extensive research has provided guidelines and tips on being a good reviewer based on interviews with editors and individual reviewing experience. For example, high-quality review

feedback was commonly considered to be clear, constructive, professional, and actionable [22, 93]. Moreover, prior research also suggested that reviewers should carefully organize their feedback, such as beginning with a brief summary and differentiating between major and minor issues [22, 37]. Interestingly, the CSCW Student Reviewer Mentoring Program [3] provided students with guidance in writing good reviews. However, previous work also identified potential problems in peer review processes, such as biased, inadequate, and delayed review [44, 49, 84]. As for guaranteeing and improving the review quality, editors play a key role in the review process in terms of selecting and guiding reviewers and assessing their feedback [49, 84]. Additionally, researchers have also compared different forms of peer review (*e.g.*, double-blind review and open review) and discussed the advantages and disadvantages of each review form [84, 89, 106].

***Feedback on student writing.*** Feedback has long been seen as critical to revising students' written productions and improving their writing skills in many different fields of education [9, 34, 41]. A substantial body of research has investigated various different aspects of feedback and their impact on feedback implementation [36, 41, 109, 110]. For example, students were motivated more by content-based feedback than form-focused feedback [105], and early evaluation of form could discourage students from further revising [66, 78]. As for the structural components of feedback, the inclusion of an explanation or a suggestion in feedback increased students' willingness to respond to the feedback [36, 109]. In terms of feedback senders, teacher feedback was generally perceived as more useful than peer feedback and had a higher chance to be incorporated into revisions [110]. However, prior studies also revealed that training helped to improve the effectiveness of peer review feedback [70]. There have also been a number of studies that suggested best practices for response to student writing [31, 61], *e.g.*, giving clear and text-specific feedback and focusing on organization issues before grammar issues. Moreover, many modern computer-based tools were developed to facilitate the feedback-giving process, such as PeerStudio designed by Kulkarni *et al.* [57] to enable rapid feedback from peers.

***User feedback on software.*** User feedback plays an important role in creating and maintaining successful software products; it helped developers to identify features and bug fixes for the next releases to ensure user satisfaction and software reliability [23, 55, 75]. The commonly used communication channels between users and developers include bug tracking system (*e.g.*, Bugzilla) [18] and mobile applications (*a.k.a* App) stores (*e.g.*, Google Play) [18, 75], and previous work has extensively studied user feedback collected from these two sources. Bug reports contain a mixture of structured and unstructured content, and a variety of information (*e.g.*, steps to reproduce and stack traces) are expected to be included to help developers to understand the problem [8, 59]. However, many bug reports were poorly written [8, 59]. Prior studies have reported that there was a huge information mismatch between what developers need and what bug reporters supply [14, 18, 59], which often resulted in delayed processing and even ignorance of bug reports [19, 42]. To improve the quality of bug reports, some studies designed automated methods to detect missing information from and provide summarization of bug reports [18, 83]. As for App reviews, prior studies analyzed the content and uncovered different feedback types and their frequency and concurrency patterns [55, 75]. Researchers also indicated that App developers were struggled with identifying relevant and useful information from a large volume and noisy reviews [23, 98]. Therefore, various techniques have been proposed for automatically classifying, summarizing and prioritizing user feedback [23, 74, 98].

The above studies provide interesting insights into the challenges and practices in three typical feedback contexts. Yet, little is known whether or not such challenges and practices should also

---

[3]https://cscw.acm.org/2019/volunteer-mentors.html

be applied to OSS contribution. We complement existing work on feedback-based revisions by studying the PR review and revision process.

## 3 STUDY DESIGN

In this study, we aim to achieve an in-depth and comprehensive understanding of repeated revisions to PRs. To achieve our research goal, we address the following research questions.

> **RQ1.** How do reviewers request changes to PRs?
>
> **RQ2.** When are submitters more willing to make requested changes?
>
> **RQ3.** What revisions could have been avoided?
>
> **RQ4.** Why are repeatedly revised PRs not merged?

To answer our RQs, as shown in Figure 2, we followed a mixed-methods approach that consisted of two parts at the high level, *i.e.*, *manual analysis*, as described in Section 3.1, and *developer survey*, as described in Section 3.2. Findings from the two sources complemented and validated each other, thereby deepening and broadening our understanding of the phenomenon of repeated PR revisions.
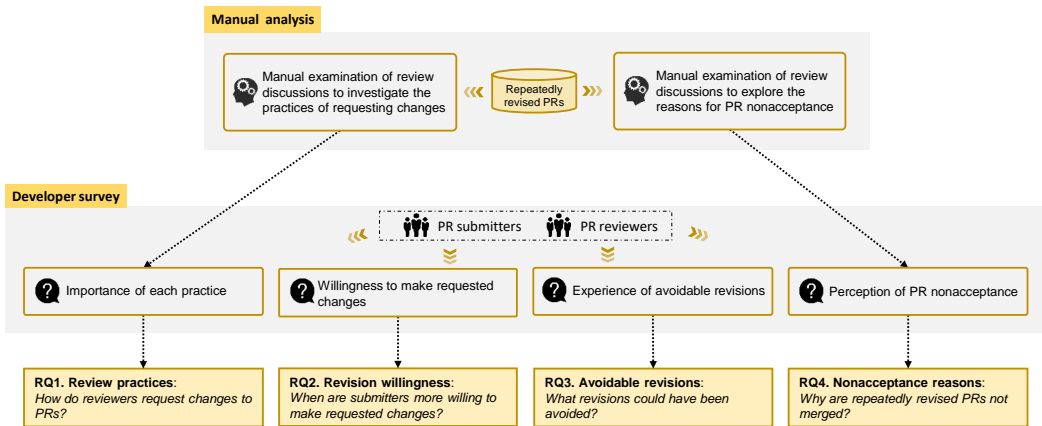


Fig. 2. Overview of methodology design.

## 3.1 Manual analysis

In this section, we present the studied projects, the repeatedly revised PRs collected from them, and the method used to analyze the collected data.

***Studied projects.*** We studied on five popular OSS projects hosted on GitHub, as shown in Table 1. To compose a sensible sample from the noise [53, 71], we focused on popular GitHub projects to avoid including inactive, personal projects and to ensure the quality of the dataset. Specifically, we used the number of stars as the proxy for project popularity [10, 11], as done in previous studies [80, 95, 107, 114]. We first obtained the list of the top 1,000 popular repositories hosted on GitHub, which had the most number of stars. From these repositories, we identified projects adopting the PR model for software development and selected 5 of them for manual analysis. The selected projects were mature and had years of development history that can be used for
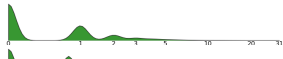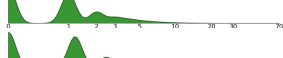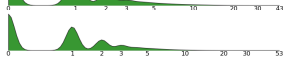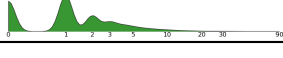
research. Moreover, these five projects used different programming languages, which increased the generalizability of our research.

Table 1. Studied projects.

| Project | Language | Created_at | #Star | #PR |
|---------|----------|------------|-------|-----|
| Rails | Ruby | 2008-04-11 | 36,330 | 13,820 |
| Django | Python | 2012-04-28 | 24,965 | 29,729 |
| TensorFlow | C++ | 2015-11-07 | 37,030 | 18,409 |
| Elasticsearch | Java | 2010-02-08 | 10,550 | 11,915 |
| Angular | TypeScript | 2014-09-18 | 22,595 | 47,522 |

***Repeatedly revised PRs.*** We collected the review data of the five projects via the GitHub official API [4]. First, we obtained all PRs of each project. Next, for each PR, we obtained events occurring in PR reviews and identified revisions as follows: all events were checked in chronological order, and a new revision was defined as a revision event that occurred after a non-revision event. Revision events include the `committed` event (a new commit is added to a PR) and the `head_ref_force_pushed` event (the head branch of the PR is force-pushed). Non-revision events are all other events except revision events, such as the `commented` event (a comment is posted on the PR). Since the first `head_ref_force_pushed` event recorded in the five projects' repositories was created on August 19, 2014, we studied PRs submitted after that day. Then, we calculated the number of revisions to each PR. In each project, PRs for which the number of revisions was three standard deviation higher than the mean value in the project were considered to have been repeatedly revised. Table 2 presents the statistics of counts of revisions to PRs in and the number of repeatedly revised PRs selected from each project. In total, we collected 1,649 repeatedly revised PRs from the five projects.

Table 2. Statistics of counts of revisions to PRs in each project.

| Project | min | max | mean | median | stdev | Distribution | #RRPRs* (% of all PRs) |
|---------|-----|-----|------|--------|-------|--------------|------------------------|
| Rails | 0 | 31 | 0.79 | 0 | 1.52 | | 239 (1.58%) |
| Django | 0 | 70 | 1.27 | 1 | 2.05 | | 158 (1.58%) |
| TensorFlow | 0 | 43 | 1.28 | 1 | 2.03 | | 279 (1.88%) |
| Elasticsearch | 0 | 53 | 1.21 | 1 | 1.89 | | 696 (2.11%) |
| Angular | 0 | 90 | 1.93 | 1 | 3.24 | | 276 (1.69%) |

*\* RRPRs: Repeatedly Revised PRs*

***Data analysis.*** Following card sorting methods [117], we manually analyzed historical review data of repeatedly revised PRs to answer the research questions. To investigate the code review practices for requesting changes to PRs (*RQ1*), the first two authors of the paper applied open card sorting to analyze a sample of review comments of repeatedly revised PRs. They constantly coded randomly selected review comments and stopped when saturation of findings [32] was

---

[4]https://docs.github.com/en/rest

reached with a criterion of 5, *i.e.*, until no new insights were obtained for 5 consecutive comments. Finally, they analyzed 51 comments and found a set of recurrent practices adopted by reviewers in requesting changes to PRs. Finally, the identified practices were discussed by all authors.

To explore the reasons for nonacceptance of repeatedly revised PRs (*RQ4*), we first collected repeatedly revised but unmerged PRs. We automatically selected all closed PRs according to their status. Since a closed PR might be either merged or unmerged [53], we then manually checked each closed PR to determine whether it was in fact unmerged. In total, we identified 149 unmerged PRs. To examine the reasons for PR nonacceptance, we performed a closed card sorting according to the categories defined by Gousios *et al.* [38]. As shown in Table 3, we organized the reasons into three groups. When a PR could not be classified into any exiting category, a new category was added. After all unmerged PRs were classified, we measured the frequency of each reason.

Table 3. The categories of reasons for PR nonacceptance defined by Gousios *et al.* [38]

| Group | Reason | Description |
|---|---|---|
| PR *per se* | Incorrect | The implementation is incorrect, missing or not following project standards |
| | Superfluous | The PR does not solve an existing problem or add a feature needed by the project |
| | Tests | Tests failed to run |
| | Process | The PR does not follow the project conventions for sending and handling PRs |
| Inter-PR | Superseded | A new PR solves the problem better |
| | Conflict | The feature is currently being implemented by other PR or in another branch |
| Project | Obsolete | The PR is no longer relevant, as the project has progressed |
| | Duplicate | The functionality had been in the project prior to the submission of the PR |
| | Deferred | Proposed change delayed for further investigation in the future |

## 3.2 Survey of developers

In this section, we describe the survey design and the participants and their responses.

***Survey design.*** In our study, we designed two surveys: one for PR submitters and one for PR reviewers, which are available online [1]. We began each survey with an introduction and a short explanation of our research. Then, we asked the participants about their OSS development experience. Next, we asked several questions corresponding to our research questions. The questions were single-choice, multiple choice, Likert scale, and open-ended questions. At the end of each survey, we asked the participants to freely provide any other information that they thought was important for our research. Table 4 lists the survey questions that were motivated by research questions .

The question *Q1* was designed for both PR submitters and reviewers to validate the practices for requesting changes to PRs extracted in manual observation (**RQ1**). Survey participants were asked to rate the importance of each practice with a five-point Likert scale (*i.e.*, *"Very unimportant"*, *"Unimportant"*, *"Neither important or unimportant"*, *"Important"*, and *"Very important"*). We also allowed participants to report other practices which they believed are important when reviewers ask PR submitters to perform PR revisions.

To explore developers' willingness to perform requested changes to their PRs (**RQ2**), we first asked submitters the question *Q2* about how many revision rounds they are willing to undergo before their PRs get merged. For data triangulation, we also asked reviewers the same question to collect their observations about after how many rounds of revisions it became hard to ask

Table 4. The list of main survey questions (the questions marked "∗" were only used in the submitter survey).

| # | RQ† | Question text | Answer choices |
|---|-----|---------------|----------------|
| Q1 | RQ1 | Please rate the importance of the following review practices when reviewers request changes to PRs | [Very unimportant, Unimportant, Neither important or unimportant, Important, Very important] |
| Q2 | | How many rounds of revisions are submitters willing to undergo before a PR gets accepted? | [0, 1, 1-3, 3-5, 5-10, always] |
| Q3* | RQ2 | How much do you agree with the following statements about the projects or PRs for which you are willing to perform continuous and repeated revisions to your code? | [Strongly disagree, Disagree, Neither agree or disagree, Agree, Strongly agree] |
| Q4* | | In which round of revision do you expect the following changes to be requested in repeated revisions? | [The first revision, The earlier the better, The later the better, The last revision, Whichever round] |
| Q5 | RQ3 | Based on your experience, what kind of revision rounds could have been avoided? | |
| Q6 | RQ4 | If any, why was a PR unmerged after repeated revisions? | The categories defined by Gousios *et al.* [38] |

† *This column refers to the research question that motivated the question.*

submitters for more work. Second, we asked submitters a Likert-scale question *Q3* about the extent to which they were willing to perform repeated revisions to their code in the given contexts.

To populate the choices, we reviewed previous studies that examined the factors affecting developers' continuous participation in OSS projects [60, 62, 69, 80, 81]. The examined factors can be split into three categories, including personal factors related to developers (*e.g.*, no time [80]), macro-level factors related to projects (*e.g.*, unwelcoming community [81]), and micro-level factors related to PRs (*e.g.*, reviewer unresponsiveness [62] ). In the survey, we deliberately chose to evaluate macro- and micro-level factors as reviewers can hardly motivate submitters by controlling personal factors. Additionally, as commercial participation in OSS continues to grow [85], we also evaluated the effect of getting paid on developers' contribution willingness. We also allowed submitters to report other important factors affecting their willingness that were not covered by the provided choices. Third, in a Likert-scale-like question *Q4*, we asked submitters to rate the priority of requesting different types of change. The ranges of priority included *"The first revision"*, *"The earlier, the better"*, *"The latter, the better"*, *"The last revision"*, and *"Whichever round"*. The types of change were populated with the taxonomy defined by Panichella *et al.* [76], as shown in Table 5. We chose this taxonomy for study because it was the latest research result at our examination time and was built on code review data of OSS projects. Additionally, we made small adaptions to the taxonomy to *i)* remove all too-specific subcategories in each theme, *ii)* reorganize categories in the 'Other' group into two general themes, and *iii)* add two new themes (marked with an asterisk (∗)) emerging in a closed card sorting [5, 117] of a sample of review comments in our dataset.

By asking RR submitters and reviewers the question *Q5*, we obtained their firsthand experience of avoidable revisions (**RQ3**). The question was an open-ended question and there were no specified answer choices. Survey participants openly provided their opinions about what kinds of revisions could have been avoided based on their experience. To group answers into categories, we manually analyzed the response text using the open card sorting approach discussed in Section 3.1 and identified common themes.

We asked both PR submitters and reviewers a multiple-choice, multiple-selection question *Q6* about the reasons for nonacceptance of repeatedly revised PRs (**RQ4**). The choices were populated with the results of manual examination in Section 3.1, and the survey participants were asked to select all the reasons that they agreed with according to their experience. The participants were also allowed to write other important reasons that we did not provide.

***Participants and responses.*** We recruited potential participants from GitHub projects. In addition to the 5 projects listed in Table 2, we additionally included 15 projects which were randomly selected

Table 5. The taxonomy of change types (new change types are marked with an asterisk (*))

| Group | Theme | Description[6, 64, 76] |
|---|---|---|
| Perfective | Documentation | documentary deficiencies in the program text |
| | Style | code-formatting without an effect on the compilation results |
| | Structure | problems related to code organization and solution approach |
| Corrective | Resource | mistakes made with data, variables, or other resources |
| | Logic | logical defects in control flow, comparisons, computations, *etc.* |
| | Interface | mistakes made when interacting with other parts of the software |
| | Check | defects in checking the value of variable, function-call and user input |
| | Larger defects | defects made with completeness, GUI, *etc.* |
| Other | Configuration | changes in the configuration |
| | Commit | changes in the commits (*e.g.*, rewriting a commit message) |
| | Rebase* | rebasing the PR to the latest main branch |
| | Unnecessary* | reverting unnecessary modifications |

from the top popular projects described in Section 3.1 and covered a wider range of programming languages. The complete list of the 20 projects is publicly available [1]. From these projects, we targeted developers who had fresh experience with PR reviews and revisions. Specifically, we first collected the 100 most recent closed PRs undergoing at least one revision from each project. Then, we sent the submitters and reviewers of the collected PRs emails containing a link to our online survey. In total, we successfully sent 1,439 invitations, including 1,171 to contributors and 268 to reviewers. The surveys ran for two weeks. Ultimately, we received 157 responses, among which 131 are from submitters and 26 from reviewers, achieving a response rate of 10.9%. Table 6 presents demographic information about the participants, which is similar to that reported in recent studies of OSS contribution and review [24, 25, 35, 46, 81, 95].

Table 6. Demographic information of survey respondents.

| Information | Submitters | Reviewers |
|---|---|---|
| OSS development experience in year? | <1:*6%*; 1-3:*25%*; 3-5:*20%*; 5-10:*19%*; >10:*30%* | <1:*0%*; 1-3:*8%*; 3-5:*27%*; 5-10:*35%*; >10:*30%* |
| Average number of PRs per month? | not sure: *25%*; 1-3:*39%*; 3-5:*7%*; >5:*29%* | - |
| Number of maintained OSS projects? | - | not sure:*0%*; 1-3:*54%*; 3-5:*23%*; >5:*23%* |

## 4 REVIEW PRACTICES

*RQ1: How do reviewers request changes to PRs?*

In this research question, we aimed to identify code review best practices for requesting changes to PRs. Section 4.1 describes the practices identified in manual observation, and Section 4.2 presents developers' evaluation on the extracted practices .

### 4.1 Review practices

Table 7 reports the taxonomy obtained in the open coding process. In total, we found fourteen code review practices for requesting changes to PRs, and organized them into six categories. For each practice, the table also lists the number of corresponding review comments from open coding.

Table 7. Code review practices for requesting changes to PRs.

| Category | Review practice | #Comments |
|---|---|---|
| Explaining *"Why"* | $\mathcal{P}$1  Explaining the reasoning behind requested changes | 8 |
| Presenting *"What"* | $\mathcal{P}$2  Inspecting high-level issues before details | 2 |
|  | $\mathcal{P}$3  Itemizing the requested changes | 2 |
| Suggesting *"How"* | $\mathcal{P}$4  Directing the submitter to a useful reference | 6 |
|  | $\mathcal{P}$5  Pointing out the change location | 2 |
|  | $\mathcal{P}$6  Implementing and providing the needed changes | 2 |
| Considering *"Who"* | $\mathcal{P}$7  Assigning more appropriate reviewers | 4 |
|  | $\mathcal{P}$8  Multiple reviewers double-checking the PR | 1 |
| Tracking *"When"* | $\mathcal{P}$9  Notifying the submitter of the review feedback | 5 |
|  | $\mathcal{P}$10 Reminding the submitter of missed comments | 6 |
|  | $\mathcal{P}$11 Scheduling the next round of review | 4 |
|  | $\mathcal{P}$12 Informing the submitter of upcoming acceptance | 9 |
| Social encouragement | $\mathcal{P}$13 Giving compliments/thanks | 9 |
|  | $\mathcal{P}$14 Requesting the changes courteously | 10 |

In the rest of this section, we describe the practices in detail. To help clarify the results, we include typical comments and the key words/phrases in some long sentences appear underlined.

*A) Explaining "Why".*

$\mathcal{P}$1 ***Explaining the reasoning behind requested changes.*** As indicated by The Golden Circle [90], good leaders and organizations inspire people to action by first explaining the purpose. Prior studies have highlighted the importance of indicating the rationale of the code change in code reviews [25, 26]. As for PR revisions, we found that reviewers tend to provide an explanation of why a PR needs to be revised, which helps PR submitters understand the necessity of the requested changes. Usually, reviewers pointed out the flaws in current implementation or underscored the advantages of a new implementation, for example:

> 💬 *In the case of DateTimeField should use timezone.now() instead. If we don't and USE_TZ=True the default datetime used on column creation will be in the system timezone instead of UTC.*

More importantly, when there were different alternative solutions, reviewers made an explicit comparison and discussed the pros and cons of each alternative for their decision-makings. Below is a good example.

> 💬 *I think that we should generate it for options that generate engine. Otherwise it will break as @xxx mentioned. The other option would be to always generate it, but add nice warning if there is no engine when you try to run it and also add option to be able to skip it. But the first option seems much simpler.*

*B) Presenting "What".*

$\mathcal{P}$2 ***Inspecting high-level issues before details.*** Requesting changes to PRs is principally about telling submitters what problems they need to solve. The problems in a PR may be related to implementation direction at a high level or coding style at a low level. Some reviewers were accustomed to first examining PRs at a high level (below is an example). Focusing first on the

significant aspects can avoid causing unnecessary work, *e.g.*, a piece of code was formatted in a certain revision, but in a subsequent revision, it was removed due to structural changes.

> 💬 *Thanks! A few high-level comments:*
>   - *Is the folder name '`keras_examples_benchmarks`' better?*
>   - *No need of new BUILD and init files in the new folder. Let's keep the folder hold the models and benchmark tests only.*
>   - `benchmark_util` *can be in the main folder, as it's also used by* `keras_cpu_benchmark`, *and might be extended with more utils later.*
>   *Will review the code details after the above changes are made.*

$\mathcal{P}$3 ***Itemizing the requested changes.*** In a code review, reviewers might identify multiple issues that need to be addressed. When adding a comment to the PR, some reviewers listed the issues one by one to make the comment clear and concise, as shown in the above comment. Since communication is a major challenge in distributed collaboration [52, 87], posting itemized text instead of plain text may help submitters quickly and clearly catch the main points. Below is another good example:

> 💬 - *rename* `NgZone` *to* `ZoneJSNgZone`.
>   - *Create new abstract class* `NgZone` *to be an injection token.*
>   - *have* `ZoneJSNgZone` *and* `NoopNgZone` *extends* `NgZone`.

### C) Suggesting "How".

$\mathcal{P}$4 ***Directing the submitter to a useful reference.*** Reviewers might help PR submitters with information on how to perform the requested changes. One common type of help from reviewers was providing a useful reference to guide the revision. The suggested reference could be a similar example or relevant guidelines in the project, as shown in the following examples:

> 💬 *To add the configuration just look what was did here [a clickable file path].*

> 💬 *Please use hanging indentation due to the coding-style [a clickable hyperlink].*

$\mathcal{P}$5 ***Pointing out the change location.*** Another kind of help from reviewers was suggesting where the revision should take place. Such information not only saves submitters' time spent determining change location but also reduces reviewers' waiting time for the expected changes. In practice, the suggested locations can be file level or line level. As examples, we have the following comments:

> 💬 *@xxx are you up for adding that? It should probably be added in the serialization documentation and in the Active Job Basics guide.*

> 💬 *Also, a mention in the release notes is needed – I guess in the 'Backwards-incompatible changes' section.*

$\mathcal{P}$6 ***Implementing and providing the needed changes.*** Sometimes, reviewers offered more direct help by implementing the patch themselves. More specifically, we observed three different ways that reviewers provided implemented changes, as listed below.

*(i)* Reviewers used the "Suggested changes" feature [16] provided by GitHub, which allows submitters to directly apply the changes suggested by reviewers via certain online clicks.

*(ii)* Reviewers implemented the changes in their own branch and asked the submitter to pull the implementation, as indicated in the following comment:

> 💬 *Would you consider to cherry-pick my commit to complete this pull request?*

*(iii)* Reviewers directly modified the PR if the project maintainers had been given the corresponding permission [5], as a reviewer commented:

---

[5]https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/allowing-changes-to-a-pull-request-branch-created-from-a-fork

💬 *Thanks for updates. I squashed commit and push minor fixes to tests. We're quite close but I have few questions/concerns: xxx.*

The first way was commonly used for small changes, such as fixing typos and code formatting, and the other two ways were usually used for more complicated changes.

*D) Considering "Who".*

𝒫7 **Assigning more appropriate reviewers.** Popular and large OSS projects are usually maintained by a team rather than a single developer. Team members within a project can have different technical backgrounds and expertise [77]. When reviewers lacked the expertise required to inspect a modified component, they would invite more appropriate reviewers who could provide more accurate and quick feedback. As examples, we have the following comments:

💬 *@xxx: maybe you would be interested as I'm sure this will require changes in pytest-django.*

💬 *@xxx is there a better way to accomplish this without reaching into an ivar?*

𝒫8 **Multiple reviewers double-checking the PR.** As indicated by Linus's law (aka. *"given enough eyeballs, all bugs are shallow"*) and previous quantitative studies [67], high reviewer participation is beneficial to software quality. Therefore, even though some reviewers had already left comments on a PR, they chose to invite more reviewers to double-check it (as shown in the comment below). This is similar to the regulations in some projects which require that each PR be approved by two or more maintainers before getting merged.

💬 *LGTM. @xxx @xxx for final review?!?.*

*E) Tracking "When".*

𝒫9 **Notifying the submitter of the review feedback.** When providing feedback, reviewers usually used "@" mentions [54] to ping submitters and remind them of the posted comments. Additionally, some reviewers used the "Request changes" feature [6] provided by GitHub. Reviewers in GitHub can add inline comments to a specific part of the change or general comments on the entire PR. When submitting inline comments, reviewers can choose to send a "Request changes" reminder. In addition, reviewers have a chance to summarize inline comments and highlight the important points in an automatically generated general comment, as shown in the following example:

💬 *I left some nits and suggestions. One change in MethodHandlers I think we do need to make though.*

Interestingly, one participant in our survey reported the usage of the "Assignee" feature [7] in notification: *"On GitHub, I use the 'Assignee' feature to designate 'who is blocking the PR' or 'on whom is the PR waiting'. Is it waiting on the reviewer to review or the author to implement changes"*. And the participant also suggested to extend the feature to support: *"ping/bump the blocker. E.g., reminder emails and notifications ('this PR is waiting on you for 3 days')"*. Actually, some GitHub projects have achieved a similar functionality using labels [8], such as the label *"stat:awaiting response"* in the Tensorflow project. As suggested, GitHub can associate labels with additional actions, *e.g.*, sending customized reminder emails, that could be automatically triggered by label assignment.

𝒫10 **Reminding the submitter of missed comments.** In a review, reviewers may provide more than one comment, whether general or inline. However, PR submitters may have ignored certain comments in preparing a revision. It is especially prominent if different reviewers left a large number of comments on different aspects of the PR. When reviewers checked the new revision

---

[6]https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/reviewing-proposed-changes-in-a-pull-request
[7]https://docs.github.com/en/github/managing-your-work-on-github/assigning-issues-and-pull-requests-to-other-github-users
[8]https://docs.github.com/en/github/managing-your-work-on-github/managing-labels#about-labels

and found that some issues remained unaddressed, they reminded the submitter of the missed comments, as observed in a comment:

💬 *This is still not done.*

𝒫11 **Scheduling the next round of review.** Given the great number of PRs and limited energy of reviewers, reviewers may not be able to give immediate feedback on a new revision. Additionally, reviewers may have time to inspect only part of a big PR at a time [12], and the remaining changes then must be left for the next review. Since a lack of responsiveness from reviewers can demotivate contributors from engaging [60, 62], making reviewers' status and availability public might help to prevent submitters' negative feelings. According to our observations, some reviewers scheduled the times when they would conduct another review on a PR. As examples, we have:

💬 *awesome, it's really close, will review again Friday.*

💬 *Over all I think this API looks good. I'll take a closer look at the tests tomorrow.*

𝒫12 **Informing the submitter of upcoming acceptance.** When a PR needs only one more revision before being merged, reviewers liked to inform the submitter of upcoming acceptance. This may act as an incentive for the submitter to complete the PR that is almost done in a timely manner. As an example, we have:

💬 *New version looks good to me. Just a few minor comments and we should be good to merge.*

*E) Social encouragement.*

𝒫13 **Giving compliments/thanks.** Code review is an inherently social activity. Reviewers assess the quality of proposed changes and interact with submitters to discuss concerns and potential defects. During the review process, submitters can have the fear of being rated and nonacceptance [39]. Social encouragement from reviewers can relieve submitters' social barriers and motivate them to participate [45]. Therefore, reviewers were accustomed to sending compliments/thanks for PRs and new revisions, as a reviewer commented:

💬 *@xxx Thank you for your changes, this looks good. Would you mind adding a test for the new behavior?*

𝒫14 **Requesting the changes courteously.** The sustainability of community-based OSS projects depends on the contributions of voluntary developers. Previous studies [21, 81] have found that the level of politeness of review comments has a strong impact on contributors' willingness and responsiveness. Thus, reviewers usually proposed change requests courteously, as shown in the following comment:

💬 *Maybe something along the lines of: Subclass of* `ActiveModel::Errors` *that properly includes association errors on the foreign key. may be better. What do you think ? And could you wrap these documentation lines around 80 chars please?*

---

**Observation 1**

*We identified 14 code review practices for requesting changes to PRs in terms of explaining the reasoning, presenting the problems (e.g., inspecting high-level issues before details), suggesting the approach (e.g., pointing out the change location), inviting relevant reviewers (e.g., assigning reviewers with related expertise), tracking PR progress (e.g., informing the upcoming acceptance), and social encouragement (e.g., requesting the changes courteously).*

---

## 4.2 Validation from developers

We validated the results of the manual analysis with a survey. Figure 3 shows the perceived importance of the extracted code review practices. For each practice, we present submitters' perceptions (left-hand subplot) and reviewers' perceptions (right-hand subplot) in the same row. In addition to the frequency of responses, we append the aggregated score result (*i.e.*, median) for each bar. The score ranges from 1 to 5, corresponding to the five answer options, as indicated by the legend of the figure.
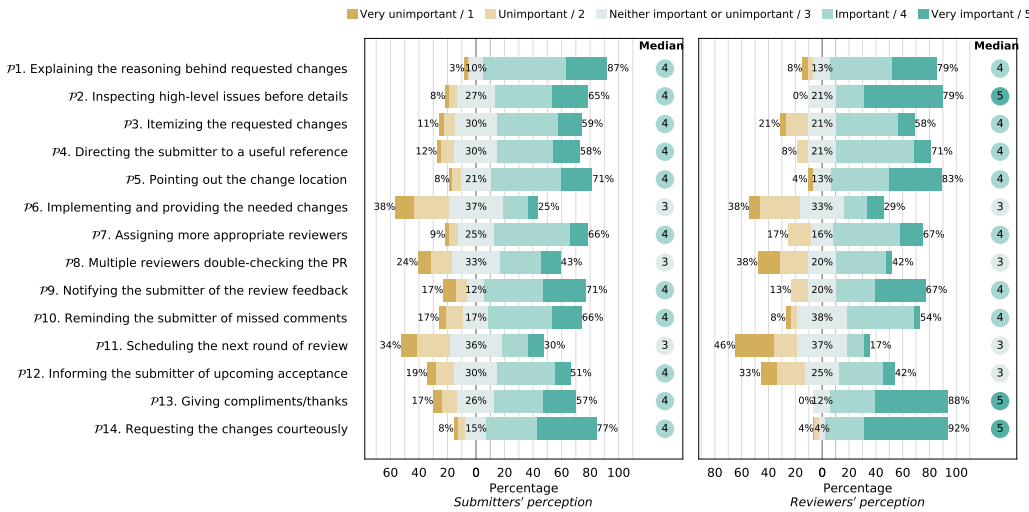


Fig. 3. Responses to the 5-point Likert-scale question for the importance of code review practices. Left-hand subplot shows responses from submitters, and right-hand subplot shows responses from reviewers.

We can see that most practices were considered important (median > 3) by both PR submitters and reviewers. Three practices, including $\mathcal{P}6$, $\mathcal{P}8$, and $\mathcal{P}11$, were considered neither important or unimportant (median = 3) by both submitters and reviewers. In addition, reviewers held neutral opinions on practice $\mathcal{P}12$. The importance of $\mathcal{P}6$ was confirmed by less than 30% of submitters and reviewers. This indicates that while changes provided by reviewers can facilitate PR review by reducing the wait time for the next revision, developers have reached a general consensus that reviewers' main responsibility in PR revision processes is to provide feedback rather than solutions. The different roles and responsibilities of reviewers and submitters are usually documented in projects' contributing guideline [9,10] and were also commonly described in prior studies [12, 38]. Regarding $\mathcal{P}8$, 24% of submitters and 38% of reviewers considered it unimportant; this can be explained by that developers care about the heavy workload of reviewers and think it is unrealistic to require that each change request, especially for small or trivial changes, be double-checked by multiple reviewers. In terms of $\mathcal{P}11$, only 30% of submitters and 17% of reviewers perceived it as important, which was contrary to our expectation. We speculate that although telling submitters the time of the next review can make reviewers' availability clear, it provides limited benefits in terms of bringing about the currently requested changes.

---

[9]https://edgeguides.rubyonrails.org/contributing_to_ruby_on_rails.html
[10]https://github.com/angular/angular/blob/master/CONTRIBUTING.md

Interestingly, when further comparing the frequency of responses between submitters and reviewers, we observe perception gaps between submitters and reviewers in terms of perceived importance of practices. From submitters' perspective, some practices were overvalued and some undervalued by reviewers. To check whether the difference in developers' perceptions was statistically significant, we applied Mann-Whitney-Wilcoxon(MWW) test (with the $p$-values adjusted using the Benjamini-Hochberg (BH) method [7]). Following the commonly used confidence level of 95% ($p < 0.05$), we observed a significant difference in all comparisons with the adjusted $p$-values=0.013.

**Overvalued practices.** Even though both submitters and reviewers agreed that reviewers should kindly and appreciatively interact with submitters ($\mathcal{P}$13 and $\mathcal{P}$14), reviewers supported these more strongly (medians: 5 vs. 4). In particular, no reviewers considered giving submitters compliments/thanks unimportant. Similarly, while a submitter explicitly suggested that *"high-level mistakes should be remarked as soon as possible"*, reviewers generally expressed stronger agreement than submitters on the importance of first inspecting high-level issues, *i.e.*, $\mathcal{P}$2 (medians: 5 vs. 4). For two practices in terms of providing suggestions on how to perform requested changes ($\mathcal{P}$4 and $\mathcal{P}$5), although submitters and reviewers had the same aggregated score of 4, a greater percentage of reviewers than submitters provided positive votes ($\mathcal{P}$4: 71% vs. 58%; $\mathcal{P}$5: 83% vs. 71%).

**Undervalued practices.** The importance of justifying the requested changes ($\mathcal{P}$1) received the most positive support from submitters: 87% considered it important and only 3% considered it unimportant. Compared to submitters, a smaller percentage (79%, although the value itself was not small) of reviewers expressed a positive attitude towards the importance of $\mathcal{P}$1. Considering that developers thought highly of $\mathcal{P}$1, future research should further investigate the potential challenges faced by developers when communicating about the reasoning behind requested changes. Compared to reviewers, a greater proportion of submitters recognized the importance of all four practices related to tracking the progress of PR review and revision, *i.e.*, $\mathcal{P}$9 (72% vs. 67%), $\mathcal{P}$10 (66% vs. 54%), $\mathcal{P}$11 (29% vs. 19%), and $\mathcal{P}$12 (51% vs. 42%).

**Additional findings.** We identified an additional practice which had been mentioned in 3 submitters' free-text responses: $\mathcal{P}$15 **Referring to tool outputs instead of a manual explanation** (*e.g.*, *"Having automated tools that verify correctness or style adherence helps reviewers give more concise feedback, by referring to these instead of having to verbosely explain and motivate every change"*). The advantage of tools in providing feedback with useful context information in a pretty and automated fashion [107] helps to explain developers' mentions of this practice. OSS platforms like GitHub should consider how to better integrate various automated tools into the PR review environment and enable developers to make good use of the tool outputs in a convenient way.

> **Observation 2**
>
> *Although most of the identified practices were perceived as important by both PR submitters and reviewers (e.g., indicating the location of the code that needs to be changed), reviewers somewhat undervalue the importance of explaining the reasoning behind change requests and tracking the progress of PR review and revision. Survey participants additionally highlighted the practice of referring to tool outputs instead of a manual explanation.*

## 5 REVISION WILLINGNESS

*RQ2: When are developers more willing to make requested changes?*

In this research question, we explore the situations in which submitters are more willing to perform the requested changes to their PRs. In the following sections, we present developers' willingness with respect to revision patience, contribution context, and priority order of changes.

## 5.1    Revision patience

Figure 4 compares submitters' and reviewers' responses to the question about submitters' patience regarding PR revisions. Even though both submitters and reviewers reported that submitters never or hardly ever refused to revise their PRs (*i.e.*, 0 round), they disagreed about the number of revision rounds. While nearly 80% of reviewers observed that it became difficult to ask for more work from submitters after 5 revision rounds, only 40% of submitters reported that they could provide only 1 to 5 rounds of revision. Although more than half of submitters claimed that they were willing to respond to every change request from reviewers, this was confirmed by only 13% of reviewers. There are two possible reasons for this gap. First, submitters overestimated their patience and chose the option that gave a sense of collaboration. Second, submitters indeed had such high patience; however, they failed to act smoothly and successfully due to various problems during the collaboration process (*e.g.*, low responsiveness of reviewers and tedious process).
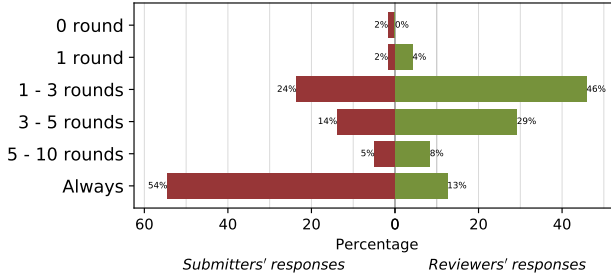


Fig. 4. Responses to the question for how many rounds of revisions submitters are willing to undergo. Left-hand responses (in Red) are from submitters, and right-hand responses (in Green) are from reviewers.

---

**Observation 3**

*Despite more than half of submitters reporting their willingness to respond to every change request from reviewers, approximately 80% of reviewers felt it is difficult to ask for more work from submitters after 5 rounds of revisions.*

---

## 5.2    Contribution context

Figure 5 presents submitters' responses to the question about the context in which they were more willing to make repeated revisions to their code. Submitters gave positive ratings (median > 3) to all provided contexts except for the context in which submitters were paid to contribute to the project ($C1$). The observation that half of submitters disagreed or strongly disagreed with the positive effect of **getting paid** on their willingness to make repeated revisions is line with the early finding that payment is a not strong motivation for contributing to OSS projects [35]. The remaining contexts can be discussed from the following two perspectives.

***Macro-level context.*** While all three macro-level contexts (*i.e.*, project characteristics), **popular projects** ($C2$), **friendly community** ($C3$), and **standard processes** ($C4$), received more than 70% positive responses, **friendly community** stands out with the largest value: 90% of submitters

mentioned that they are willing to perform repeated revisions when the project community is welcoming and friendly. This response stresses the importance of community building [12]. In practice, OSS projects usually adopt a code of conduct [100] to establish ground rules for communications among community participants. In GitHub, projects can also use robots to welcome first-time contributors [107, 108]. It would be useful to investigate how to further leverage robots to help reviewers adhere to the code of conduct during PR review processes.

***Micro-level context.*** Among the four micro-level contexts (*i.e.*, PR characteristics), it is interesting to note that the dynamics of PR review process, including **responsive feedback** ($C5$) and **helpful comments** ($C6$), were considered more motivating than the **important value** of PRs *per se* ($C7$). For example, two participants suggested: *"All change requests are welcome and good, if they are made early and if the reviewers are willing to track your progress"*, and *"If the reviewer suggested better ways of solving a problem that my PR solves"*. However, if a PR was submitted owing to a **personal need** ($C8$) to fix a problem directly affecting submitters, a greater percentage of submitters were willing to offer repeated revisions. The stress laid by submitters on reviewers' feedback highlights the importance of introducing process automation in PR reviews. Projects should broadly use automatic tools to solve trivial tasks, *e.g.*, checking CLA, building test, and examining coding style, to reduce reviewers' burden and increase their responsiveness on other, more challenging tasks [17, 92, 107]. Additionally, projects can establish a set of best practices for inexperienced reviewers to guide them in how to behave as expected and provide helpful comments.
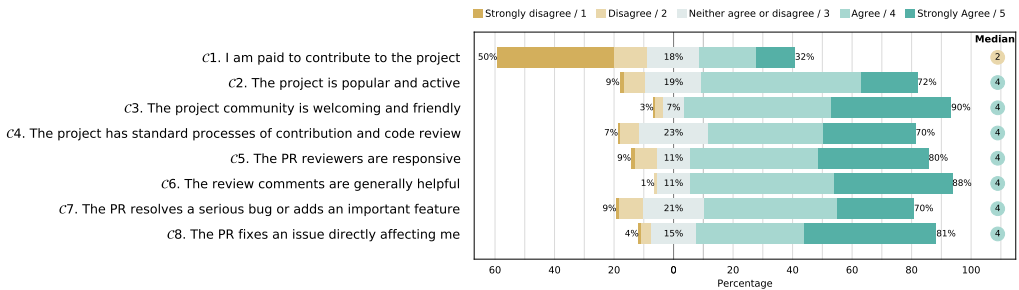


Fig. 5. Responses to the 5-point Likert-scale question regarding agreement/disagreement with the statement about types of project or PR for which submitters are willing to perform repeated revisions to their code.

***Additional findings.*** We also observed interesting ideas from submitters' free-text feedback. 5 submitters told us that they were willing to perform repeated revisions to their PRs if they can **learn and improve** from the revision process (*e.g.*, *"[if] repeated revisions allow me to learn and improve as a programmer"*, and *"mostly learning new things about writing coce"*). Additionally, 4 submitters cited **reasonable request** as a reason affecting their willingness, as shown in two quotes: *"If I find the requested changes sensible"*, and *"I am completely fine with iterating on PRs when it makes sense. But expanding the scope of a PR without the consent of the person who submitted it is not great"*. For 2 submitters, they were more willing to perform updates to a PR that has **technical defects** rather than a PR that only has superficial issues, as one said: *"Generally speaking, the only time I am willing to update a PR is when my patch is defective. If the patch works then it should be accepted. If a maintainer does not like some about the PR, then they need to fix it themselves."*

**Observation 4**

> *Both macro-level contexts (e.g., a friendly community and having a chance to learn) and micro-level contexts (e.g., helpful feedback, and solving a problem for personal needs) can increase submitters' willingness to provide repeated revisions to their PRs.*

## 5.3 Priority order of change types

Figure 6 shows submitters' opinions on the priority of requesting different types of changes. Generally speaking, corrective changes were expected to be requested earlier than perfective changes and other type of changes, as a participant mentioned *"First bugs, then logic and efficiency, then configuration then docs and linting"*.
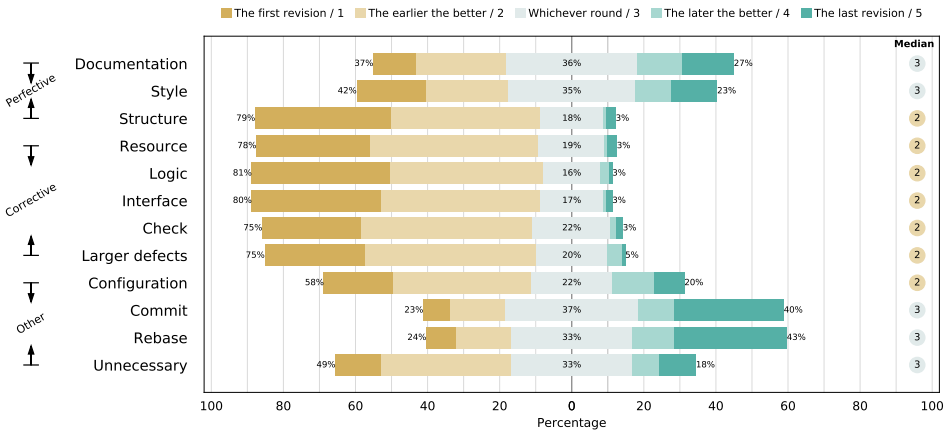


Fig. 6. Responses to the 5-point Likert-scale-like question for the priority order of requesting different kinds of changes.

In terms of perfective change types, only **Structure** was expected to be requested early (median < 3), and the other two change types did not receive overwhelmingly positive or negative responses (median = 3). Surprisingly, only 23% of submitters hoped **Style** would be requested in late revisions, and more than 40% of submitters preferred early revisions. This is somewhat inconsistent with submitters' appreciation for the code review practice of inspecting high-level issues before code details ($\mathcal{P}2$). One possible explanation is that although submitters expected reviewers to first review the significant aspects, if formatting issues were pointed out early submitters could easily avoid introducing the same kind of issues in late revisions. More interestingly, one submitter suggested that *"Style things like indention should not be requested manually but rather cause a check to fail"*.

For all five corrective change types, most submitters hoped reviewers would request them in early revisions (median < 3). This may be because corrective changes require mostly a fine-grained context, and submitters therefore hoped such changes would be requested early when the context was still fresh in their minds. Otherwise, it could become difficult for submitters to recall the details, as a survey participant mentioned: *"It's quite complicated to update PR on which I worked several months ago"*.

Regarding four other change types, submitters expected **Configuration** to be pointed out in early revisions (median < 3). A possible reason is that if configuration-related issues are not solved in a timely manner, they might hinder the operation of third-party tools, *e.g.*, CI building and

static analysis, and reduce review efficiency. Although **Commit**, **Rebase**, and **Unnecessary** all received a median score of 3, they had different response distributions. More than 40% of submitters expected **Commit** and **Rebase** to be requested in late revisions, and more than 20% of submitters chose early revisions. Especially, one submitter said that *"If a change is needed in response to highly volatile conditions, such as where the head of the master branch is, that change should be requested immediately prior to merge. Otherwise, sooner is better"*. In contrast, for **Unnecessary**, approximately 50% of submitters preferred early revisions, and only 18% chose late revisions.

> **Observation 5**
>
> *Corrective changes that generally require a fine-grained context were expected to be requested in early revisions, while changes manipulating commits and rebasing operations were expected to occur later.*

## 6 AVOIDABLE REVISIONS

*RQ3: What revisions could have been avoided?*

The purpose of this research question is to seek opportunities to reduce unnecessary revisions to PRs. Table 8 lists factors causing avoidable revisions extracted from developers' self-reported experience, which can be grouped into four aspects. For each factor, the table also reports the number of mentions among submitters and reviewers, respectively. Below, we discuss each factor in detail.

Table 8. Factors causing avoidable revisions to PRs.

| Category | Factors causing avoidable revisions | Mentions | |
|---|---|---|---|
| | | #submitters | #reviewers |
| Review feedback | $\mathcal{F}1$ Misunderstanding the PR | 2 | 1 |
| | $\mathcal{F}2$ Confusing review comments | 1 | - |
| | $\mathcal{F}3$ Conflicting advice | 2 | - |
| Review policy | $\mathcal{F}4$ Inflexible review process | 2 | - |
| | $\mathcal{F}5$ Improper order of review focuses | 3 | - |
| | $\mathcal{F}6$ Multiple early rebasing | 1 | - |
| | $\mathcal{F}7$ Unnecessary squashing | 1 | - |
| Pre-submission | $\mathcal{F}8$ Not following project conventions | - | 1 |
| | $\mathcal{F}9$ Lack of automatic formatting | 7 | 5 |
| New revisions | $\mathcal{F}10$ Incomplete revisions | - | 1 |
| | $\mathcal{F}11$ Incorrect revisions | - | 1 |

*A) Troubles with review feedback.*

$\mathcal{F}1$ **Misunderstanding the PR.** A PR might be misunderstood by reviewers, leading submitters to make invalid revisions. According to survey participants, the misunderstanding could result from both the submitter side (*e.g.*, *"poor PR Description"*) and the reviewer side (*e.g.*, *"the reviewer was rushed and failed to read the PR correctly"*). This may partially explain why submitters or reviewers often reedited the description (*i.e.*, title and body) of a PR which was already under review.

$\mathcal{F}2$ **Confusing review comments.** Previous studies showed that reviewers experienced confusion in code reviews [25, 26]. However, submitters may also experience confusing review comments and consequently conduct incorrect revisions which need to be fixed by additional revisions, as a PR submitter complained: *"a revision was made incorrectly due to the unclear/problematic review comment and I need to redo it"*. This factor is supported by evidence from our manual observation of real-world review discussions of PRs:

> 💬 *You should use 1 instead of True, sorry for misleading. If you will use True as a param than it will be automatically converted into 1.*

> 💬 *Oh sorry it's not ops:nn, it's .../python:nn.*

$\mathcal{F}3$ **Conflicting advice.** A PR is often reviewed by multiple reviewers. Unfortunately, reviewers might disagree with each other and propose conflicting advice to submitters, as a participant mentioned: *"multiple reviewers with multiple opinions on code style and latest architectural guidelines"*. In such cases, submitters might be asked to revert a prior revision (*e.g.*, *"If a change is requested, and then they request another change undoing work they asked you to do"*). Review disagreement was also observed in our manual analysis, and below is such an example:

> 💬 *[Submitter] I got the suggestion from @xxx to add this checking here. Which one should I follow?* ↘

> 💬 *[Reviewer] I'm a lot more comfortable approving changes which do not touch core tf functionality.* ↘

> 💬 *[Submitter] Sure. I reverted this changes in the new commit.*

*B) Problems with review policy.*

$\mathcal{F}4$ **Inflexible review process.** Some PR submitters reported that an inflexible code review process can result in unnecessary revisions. For example, a participant complained: *"whitespace formatting, Changelog, and other useless things. The maintainer should remove barriers and do the useless things themselves"*. Another participant suggested that *"more time is spent going back and forth between the author and reviewer than it would take for the reviewer/maintainer to implement it themself"*. Even one reviewer also pointed out that *"Sometimes, it can be simpler for the maintainer to just go directly and make the requested changes (especially when it is for minor violations, such as naming conventions, license headers, and the like). While this is a feature available in both GitHub and GitLab, I haven't seen it used to its full potential"*. Yet, we ever found cases where reviewers/maintainers did make the trivial changes themselves when merging a PR, as shown in the following comments:

> 💬 *Could you please change this to DCHEC_NE? → Never mind. I think I'll fix it internally.*

> 💬 *Hi, I already manually merged this PR at xxx since I didn't want to wait for a round-trip. I fixed the lint and other minor issues myself.*

$\mathcal{F}5$ **Improper order of review focuses.** As aforementioned above, a change request can be made at either a high level or a low level. Some submitters believed that minor changes made early might be made in vain if reviewers afterward asked for large modifications due to high-level issues (*e.g.*, *"nitpicks on code quality before the feature and general approach has been fully accepted by the core team"* and *"where important errors are not highlighted early on"*). Even worse, we observed several PRs in which the submitter performed multiple revisions according to reviewers' feedback, but reviewers finally said they could not accept the PR, as shown in the following comment:

> 💬 *@xxx thank you very much for your awesome work but we think this should not be part of the framework.*

$\mathcal{F}6$ **Multiple early rebasing.** A PR may become outdated as the main branch has progresses. Usually, reviewers asked submitters to rebase the PR before they provided another review. However, a submitter suggested that: *"multiple rebases should be avoided, just wait until it's ready to be merged, rebase once"*. To make matters worse, submitters sometimes needed to resolve merge conflicts when they conducted the rebasing operations [15]. The reason why a PR becomes outdated frequently might be the rapid development of the project or the slow feedback of reviewers.

$\mathcal{F}7$ **Unnecessary squashing.** In some projects, reviewers asked submitters to squash PR commits into one commit before merging the PR [53]. Commit squashing is generally considered beneficial to keeping the git history clean and easy to follow. Nevertheless, one participant argued that *"squashing is often redundant unless a large number of commits are involved"*.

*C) Pre-submission issues.*

$\mathcal{F}8$ **Not following project conventions.** Many projects have rules and conventions regarding the contribution criteria and processes, which are usually specified in the project guidelines. Reviewers said that some revisions could have been avoided if the submitter had strictly followed the project conventions (*e.g.*, *"adding a test should be done before any review takes place"*). This confirms the previous finding that PRs submitted by the community are not always consistent with the guidelines or practices in a project [28, 40, 118].

$\mathcal{F}9$ **Lack of automatic formatting.** Both PR submitters and reviewers stressed the usefulness of automatic tools for code formatting in reducing revisions. On the one hand, reviewers complained that some submitters did not use the automatic tools provided by the project to format their code before PR submission (*e.g.*, *"Minor nits about format and styling. This can be done automatically by an auto formatting tool"*). On the other hand, submitters complained that some projects did not provide such tools for local usage (*e.g.*, *"Some of the nit-picking around code formatting. projects should have tools, linting, etc in place to enforce project coding standards"* and *"Linting and integration tests are the reason for most of my revisions, since I cannot figure out how to run them locally prior to committing"*).

*D) Issues of new revisions.*

$\mathcal{F}10$ **Incomplete revisions.** According to the participants, some revisions were made to fix a previous incomplete revision (*e.g.*, *"the contributor didn't solve everything from a previous round"*). This was confirmed by the comment below from a real-world PR where the submitter missed a comment and responded to it in a new revision. This highlights the importance of the code review practice of checking the completeness of new revisions ($\mathcal{P}10$).

> 💬 *my bad, I totally missed that comment, going to look into that and will respond.*

$\mathcal{F}11$ **Incorrect revisions.** Another problem with a new revision is the incorrect implementation. A new revision may solve the detected issues incorrectly or result in new issues, as one reviewer said: *" One common case where there are multiple rounds is when the contributor has poor skill and/or poor idea about what they're doing, so every new revision they produce has new flaws"*. In such cases, an additional revision is required to fix that revision, as shown in the comment below:

> 💬 *Unfortunately the new changes need clang-format change too. Could you please re-apply clang-format to the files?*

To address this problem, one reviewer suggested that: *"initial rounds for PRs that aren't the correct direction could be solved by a conversation with maintainers beforehand"*.

**Observation 6**

> *We identified 11 factors leading to unnecessary and avoidable revisions that are related to reviewers' feedback (e.g., conflicting advice), code review policy (e.g., multiple early rebasing), pre-submission issues (e.g., lack of automatic formatting), and implementation of new revisions (e.g., incomplete revisions).*

## 7  NONACCEPTANCE REASONS

*RQ4: Why are repeatedly revised PRs unmerged?*

With this research question, we aim to understand why PRs that had been repeatedly revised were not ultimately merged. Figure 7 shows the frequency of nonacceptance reasons from the manual examination of historical data. We have observed two new reasons related to the social aspects of PR review, which are marked with an asterisk. **Abandoned** means the PR submitter did not respond to the last change request. **Ignored** means the reviewers did not provide feedback on the last revision. Figure 8 presents PR submitters' and reviewers' votes on nonacceptance reasons. When comparing the frequency of reasons between manual examination (Figure 7) and developers' perceptions (Figure 8), we can observe several interesting points, as discussed below.
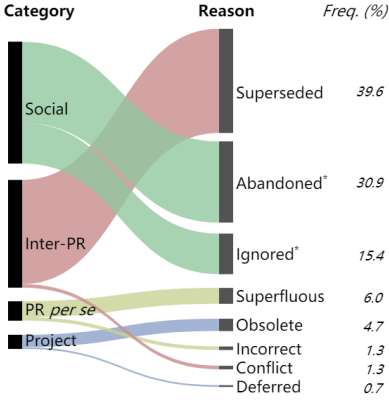


Fig. 7. The distribution of reasons for nonacceptance of repeatedly revised PRs (new reasons are marked with an asterisk (*)).
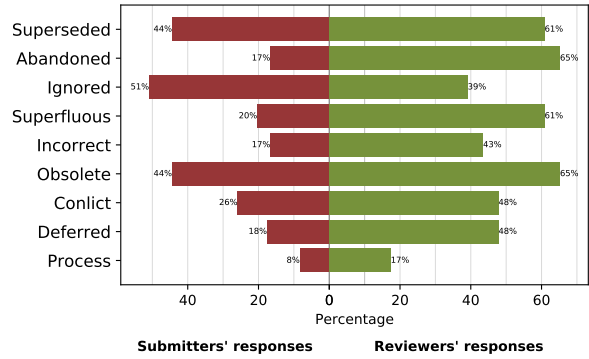


Fig. 8. Responses to the question for why PRs were unmerged after repeated revisions. Left-hand responses (in Red) are from submitters, and right-hand (in Green) are from reviewers.

***Superseded vs. Continued.*** The most frequent reason in our manual examination was **Superseded**, which was also reported to be the most common reason for nonacceptance of general PRs [38]. However, from both submitters' and reviewers' perspectives, although **Superseded** was still a common reason, it was not as outstanding as in the manual examination. Taking a closer look at the superseded PRs, we found that a majority of them were replaced by a new PR submitted by the same author. We argue that some developers might perceive these new PRs as a follow-up PR rather than a superseding PR of the original PR, consequently decreasing their votes in favor of **Superseded**. Additionally, we examined the comments explaining the reasons for replacing the original PRs and observed three main reasons: *i)* tracking the latest direction (*e.g., "At the end I preferred to create a new PR to come back to a fresh environment for further discussions"*); *ii)* cleaning up a messy review history (*"because this one was too polluted with preview comments and timeline events"*); and *iii)* splitting the PR into small pieces (*e.g., "I think a separate PR for the reference implementations and a separate PR for the tooling changes would be great"*). This reveals the gap between developers'

needs for convenient tracking and management of the record of PR discussions and the review environment provided by GitHub.

***Abandoned vs. Ignored.*** Two social reasons, *i.e.*, **Abandoned** and **Ignored**, were the second and the third most common reasons occurring in the observed PRs, respectively. This means that a significant portion of repeatedly revised PRs were unmerged due to developers' unresponsiveness. While survey participants confirmed the prevalence of these two reasons, PR submitters and reviewers expressed opposite opinions on which reason was more severe. For submitters, **Ignored** happened more frequently than **Abandoned**, but for reviewers, it was the opposite. Regardless of the party lacking a response, it is unfortunate that repeatedly revised PRs were closed due to inactivity since both parties had spent considerable effort and time in many rounds of reviews and revisions. To better understand this problem, we asked submitters/reviewers why they had abandoned/ignored a repeatedly revised PR. For PR abandonment, the majority of the cited reasons have already been reported in previous research studying abandonment of general PRs [62], and the reasons mentioned more frequently were lack of reviewers' response, no time, and lack of interest. Additionally, we found three new reasons from submitters' responses: unreasonable requests, unclear comments, and social anxiety. For ignoring PRs, the mentioned reasons were lack of reviewers' consensus, no time, unsatisfactory implementation, unsolved issues, and doubt about PR value.

***Result-oriented vs. Process-oriented.*** Our manual examination shows that only a small fraction of repeatedly revised PRs were closed due to technical issues related to either the project (**Obsolete**, and **Deferred**) or the PR (**Superfluous**, and **Incorrect**). However, these reasons had higher occurrences in developers' responses, especially in the reviewers'. In particular, both submitters and reviewers frequently mentioned **Obsoleted**, which means that the progress of project had a high chance of obsoleting a repeatedly revised PR in developers' minds. One explanation for the inconsistency may be that developers had been deeply involved in inspecting and fixing technical issues in repeated PR revisions, and consequently they tended to choose technical reasons when asked to describe PR nonacceptance.

---

**Observation 7**

*Forty percent of repeatedly revised but unmerged PRs were replaced by a new PR mainly for better tracking and management of the record of PR discussions. Although both submitters and reviewers agreed that a significant portion of PRs were unmerged due to lack of inactivity, both sides thought the other side was the unresponsive one.*

---

## 8 DISCUSSION

Based on the results described in previous Sections, we provide additional discussion and propose actionable suggestions for OSS practitioners and tool builder.

### 8.1 Main findings

***Requesting revisions is not a slam dunk.*** The results of our empirical study demonstrate that requesting revisions to PRs is a rather complex procedure that involves multi-dimensional aspects. In line with the previous findings reported in other feedback scenarios [19, 93, 109], it can be challenging to request revisions in ways that are motivating, encouraging, and clear. Being an expert in coding doesn't automatically make a developer a good PR reviewer; developers need to devote time and effort to practicing and improving their reviewing skills in order to make effective revision requests. However, the increasing number of PR submissions leads to a constant shortage of

reviewers and time has been the biggest challenge faced by reviewers [40, 112]. Therefore, relying on individuals' readiness to follow best practices for requesting PR revisions does not always work. The conflict between the complexity of making high-quality change requests and the heavy workload of PR reviewers may account for some reviewers' underestimation of certain practices (*Section 4.2*) and problematic requests (*Section 6*). Several automated methods have been proposed to reduce reviewers' workload [107, 112, 115], and researchers and tool designers should further focus on the improvement of the process and tools used to request revisions to PRs. There is an opportunity to learn from feedback system for student writing evaluation, such as scaffolding reviewers with short tips for writing meaningful revision requests [57].

***Negative impacts are magnified in repeated revisions.*** Poor reviewing practices have a negative impact on review efficiency and effectiveness [39, 60, 62]. Our findings show that the negative impact can be magnified in repeated revisions. In repeatedly revised PRs, submitters' abandonment was a dominant reason for PR nonacceptance (*Section 7*), which did not stand out in general PRs [38]. Many of our survey respondents blamed their abandonment on reviewers' improper behaviors, such as unresponsiveness, unreasonable requests, and unclear comments. Additionally, repeated revisions with several unnecessary ones caused by reviewers (*e.g.*, $\mathcal{F}5$ and $\mathcal{F}6$) may frustrate submitters and lead them to doubt reviewers' abilities and lose interest in the project [60]. This suggests that experiencing frequent collaboration issues during repeated revisions may accumulate submitters' negative feelings, thereby significantly reducing their willingness to participate.

***The social process matters.*** PR review is fundamentally a human-centric process which involves direct collaboration between the PR submitter and reviewers. Our results find that the social aspects of the collaboration process are crucial for successful PR revisions (*e.g.*, $\mathcal{F}13$, $\mathcal{F}14$, $C3$, and $C5$). This is consistent with previous studies on the social process in OSS communities [39, 94, 102]. Interestingly, although more than 40% of the surveyed submitters did not support the importance of giving compliments/thanks ($\mathcal{F}13$) in a single revision, 90% of them were willing to perform repeated revisions if the project community is welcoming and friendly ($C3$). We speculate that even though compliments/thanks do not provide concrete technical guidance in implementing reviewers' feedback, they can create a supportive collaborative environment in which PR submitters are encouraged to participate in continuous PR improvement. Considering that most of the developers are only willing to undergo several rounds of revisions (*Section 5.1*), reviewers should develop good social skills to motivate submitters in repeated revisions.

***Explanations make for revisions.*** Explaining the reasoning behind requested changes was perceived as the most important practice by submitters ($\mathcal{F}1$). Revision requests with an explanation would appear more reasonable, which can potentially increase submitters' willingness to continuously revise their PRs (*Section 5.2*). This result reinforces the evidence of the positive effect of providing an explanation in feedback on the likelihood of implementation of the feedback [47, 109]. Nevertheless, Huang *et al.* [45] presented that explanations have no effect on developers' continuous participation when they disagree with reviewers' rejection of their PRs. There are two possible reasons for the inconsistent effects of explanations. First, it is possible that explanations for PR rejection are often superficial [45], while explanations for PR revision are usually technical and improvement-orientated which seems more acceptable to developers. On the other hand, it may be a matter of effort investment. In the PR revision scenario, developers are expected to perform additional rounds of revisions, while in the PR rejection scenario, developers have to prepare and submit new PRs from the scratch. Therefore explanations have higher chances of being effective in the former scenario where developers can devote less effort to receive credit and gain peer recognition for PR acceptance [65, 91].

***The easy-to-follow principle is important to follow.*** PR review is a bi-directional process in which the PR submitter and reviewers interact with each other for PR improvement. Smooth communication between developers in the PR review process facilitates efficient PR revisions. Our findings presented that both PR submitters and reviewers considered it important to convey important information in straightforward and easy-to-follow ways ($\mathcal{P}$3 and $\mathcal{P}$5). This is in line with the previous observation of high-quality written feedback in the contexts of paper reviewing[22, 37] and issue reporting [8]. However, PR submitters and reviewers also complained about the confusing texts which led to additional unnecessary revisions ($\mathcal{F}$1 and $\mathcal{F}$2). The failure in communication may be caused by newcomers' lack of communication skills [94] or the different cultural backgrounds of developers [46, 88]. Interestingly, some developers suggested that tool outputs are easier to follow than human comments ($\mathcal{P}$15). Since the OSS community has a long tradition of applying various tools to better support both technical and social activities in developer collaboration [97, 107], the CSCW community may design more targeted solutions to assist OSS developers in providing comments that are clear and easy to follow.

## 8.2 Suggestion for OSS practitioners

Although we acknowledged that not all the identified best practices and inefficiencies in PR revisions can be ideally followed or addressed by all projects in practice, there are some points that deserve special attention from OSS practitioners.

***Requesting with an explanation.*** Our findings show that a perception gap existed between PR submitters and reviewers with respect to the perceived importance of the practice of explaining the reasoning behind requested changes ($\mathcal{P}$1). Therefore, we recommend that reviewers provide an explanation when requesting changes to PRs. The direct benefit is that it would help submitters recognize the necessity of requested changes. Another potential benefit of explaining the reasoning is facilitating knowledge sharing [2] in global and distributed collaboration. For example, PR submitters can learn from reviewers about why their implementations did not work and gain useful knowledge or skills [4, 12].

***Being clear on progress.*** Another importance perception gap existing between PR submitters and reviewers is regarding the supervision of PR progress ($\mathcal{P}$9, $\mathcal{P}$10, $\mathcal{P}$11, $\mathcal{P}$12). Reviewers were expected to be clear on the progress of PR review and revision, *e.g.*, remembering to check whether their comments had been addressed after PR revisions. Perhaps OSS projects, especially those that can obtain support from foundations or companies, can learn from journals of the academic community and assign certain "administrators" or "assistants" who are responsible for tracking the progress of PRs in a project. Team members in such roles could help to reduce the maintenance workload of PR reviewers, who could then focus on technical tasks. Additionally, some tedious and repetitive tasks could be further simplified or reduced by adopting robots [107, 108] and labels.

***Maintaining consensus on project standards.*** For growing OSS projects, the standards of processes and contributions may change over time with increases in project popularity and community diversity. Moreover, reviewers in a project may have their own strategies in reviewing. To avoid conflicting change requests that can cause unnecessary revisions ($\mathcal{F}$3), reviewers should maintain consensus on project standards. The way to achieve this objective is twofold. First, the project should document all expected review policies clearly and keeps the documentation up-to-date. In practice, despite the prevalence of contributing guidelines for submitters in OSS projects, there is a lack of a guideline for reviewers. Second, reviewers, especially new collaborators [3], should follow the documentation in code reviews to properly work with outsider submitters to reduce their barriers and challenges [39, 94]. Moreover, reviewers should be informed of the updated policies, if

any, in a timely manner. Future research can be conducted to explore the potential challenges in maintaining consensus in large-scale projects.

***Caring about size as well as rounds.*** Some projects suggested that reviewers review a bit at a time for fear of overwhelming submitters with a single big review. However, this would result in an increase in the number of PR revisions. Our results showed that a high number of revision rounds can weaken submitters' willingness to make requested changes. Particularly, one participant in our survey suggested that *"Too many revisions may make me think that the original work was incorrect, and perhaps a different approach should be reopened as a new PR"*. Therefore, restricting the size of a review but limiting the total number of review rounds is a trade-off that reviewers should be aware of. Future work can study the specific effect of review size on revision response time and total PR review time.

***Confirming before acting.*** It is inevitable that communication failures occur in code reviews due to differences in developers' backgrounds and habits, usage of various slang terms, and the talk culture of OSS communities. Before starting to make changes, submitters should confirm with reviewers that their understanding of reviewers' comments and their planned approach to address the comments is correct ($\mathcal{F}2$ and $\mathcal{F}11$). Moreover, when submitters have doubts about reviewers' comments, they should respond to confirm that reviewers have understood the PR correctly ($\mathcal{F}1$).

## 8.3 Implication for tool design

To better support collaboration between PR submitters and reviewers, there are many mechanisms and features can be implemented by tool designers to provide developers with a collaborative environment of higher transparency, automation, and intelligence. Here below are some design proposals.

***Highlighting unaddressed comments.*** In PR discussion pages, GitHub linearly displays all human comments, robots' reports, and other events (*e.g.*, change of labels and assignee) in order of creation time. For repeatedly revised PRs which tend to have an extended review history as a developer commented *"Well at least this way the PR page doesn't take 5 seconds to render"*, it is difficult to locate important and unaddressed comments. To relieve this issue, in addition to the current tab listing all timeline activities, another tab could be added to highlight unaddressed comments. If reviewers confirm that an unaddressed comment has been correctly solved by a new revision, they can mark the comment as addressed, and it will no longer be displayed in the tab. Moreover, an unaddressed comment could be automatically located to the entire discussion thread to give the context. This mechanism would not only prevent submitters from missing review comments ($\mathcal{P}10$ and $\mathcal{F}10$), but also separate ongoing discussion from the lengthy review history, which would help developers to focus on current status and reduce the chance of superseding a PR discussed in Section 7.

***Providing a review checklist.*** Although code reviews in OSS communities in general do not have criteria as strict as those for traditional code inspection [29], GitHub could still provide a review checklist in PR discussion pages to support and guide reviewers through PR review steps. The checklist would recommend the optimal order of review focus defined by the project maintainers (*e.g.*, appropriateness → implementation direction → logic → performance → code style → commit ), and reviewers would be expected to follow the checklist in order ($\mathcal{P}2$ and $\mathcal{F}5$). Each item in the checklist could be associated with rules, *e.g.*, whether the item needs to be re-examined for each new revision, whether the item should be skipped for certain types of modification, and how many of reviewers will be required to sign off on it ($\mathcal{P}8$, $\mathcal{F}1$, and $\mathcal{F}3$). This tool can also help submitters have a general understanding of the PR review process in a project and stay aware of what stage their PRs are in ($\mathcal{P}12$).

***Indicating developers' availability.*** Maintaining awareness during collaboration is important for distributed software development [43]. We recommend indicating the availability of participants in PR discussion, including *activeness* (the last time the developer visited the platform) and *busyness* (the number of tasks created/discussed by the developer in/out the project in the previous month/week). In GitHub specifically, it can be added to the popover showing developer profile that appears when the mouse is hovered over the avatar image of a discussion participant. Availability information is beneficial for flexibly manipulating the process of code reviews. For example, if reviewers/maintainers find that the PR submitter has been not around for a while or is heavily occupied by other tasks, they can implement trivial changes themselves ($\mathcal{P}6$), especially when the changes are rebasing ($\mathcal{F}6$), commit squashing ($\mathcal{F}7$), and nitpickings before acceptance ($\mathcal{F}4$). Additionally, when submitters do not hear any feedback from reviewers for a long time, availability information can lead to more informed decisions on identifying who to ask. For example, submitters can ping the reviewer with the lowest workload, who has a higher chance of responding. This may mitigate abandonment and ignorance of PRs described in section 7.

***Monitoring sentiment in comments.*** OSS collaboration involves plenty of social interaction in which developers can express various attitudes and sentiments (*e.g.*, praise, criticism, anger, and surprise) [33, 72]. Our results showed that reviewers' sentiments can affect developers' willingness to perform requested changes ($\mathcal{P}13$, $\mathcal{P}14$, and $\mathcal{C}3$). Hence, we envision a real-time tool (feasible based on off-the-shelf sentiment analysis tools [73] or popular deep learning techniques [20]) that monitors the sentiment embodied within the comment being entered by reviewers and provides suggestions on how to polish a comment before a reviewer posts it. For example, if the tool detects too much criticism and anger in a comment, it warns the reviewer about the negative sentiment. At the meantime, it provides common neutral alternatives or suggests adding positive words or emojis to neutralize the tone. Moreover, the tool can consider the number of undergone revisions and the role and contribution experience of the submitter. For example, if a PR has been revised many times or the submitter is a newcomer, the tool should suggest that reviewers use more encouraging and motivating words.

## 9 THREATS TO VALIDITY

In this section, we discuss the threats to validity of our study.

The first threat relates to the definition of repeated revisions. In the paper, we identified repeatedly revised PRs based on statistics of the dataset, *i.e.*, PRs for which the number of revisions was three standard deviation higher than the mean were considered repeatedly revised. However, as there is no consensus on the revision threshold in the literature, one can increase or decrease this threshold or use other selection criteria, leading to a larger or smaller dataset. The size of the dataset can potentially influence the outcome of manual analysis. To reduce this threat, we conducted surveys with developers, which helped us obtain more complete information.

The second threat concerns the card sorting procedure because it is a human process that is error prone and might introduce subjective bias. To mitigate this threat, manual classification was conducted jointly by multiple authors of the paper. When they disagreed with each other, they discussed until a consensus on the classification was reached. In addition, we cannot guarantee that we have covered all the topics in data classification. We have reduced this issue by setting a conservative saturation point for data classification.

The third threat relates to our findings from the survey. It is possible that some questions might have been misunderstood by the participants. Moreover, as the respondents voluntarily participated in our survey, the responses might be more skewed towards developers who have spare time and have a special focus on repeated PR revisions. To mitigate this threat, we carefully designed

the survey and encouraged responses by discussing with experienced researchers in conducting surveys, and performing pilot surveys internally.

The forth threat concerns the generalizability of our results. We studied on a set of popular GitHub projects in terms of stars. Actually, there are a great number OSS projects hosted on GitHub and other platforms and various mechanisms can be used in project selection. Therefore, we cannot assume that our findings can be generalized to all OSS projects. However, the fact that many prior studies [25, 45, 80, 95, 107] have also studied on only a sample of popular projects gives our choice some validity. Additionally, the survey sample may not be representative of all developers across all OSS projects. Nevertheless, the random selection used in participant recruitment and the similarity between the respondents' demographic and that reported in previous related studies [24, 25, 35, 46, 81, 95] helped to improve the level of participant representativeness. Further study is desirable to validate other projects and developers in and outside GitHub.

## 10 CONCLUSION

In this paper, we report an empirical study of repeatedly revised PRs. By manually investigating historical data gathered from 5 well-known OSS projects and surveying both PR submitters and reviewers, we conducted an in-depth analysis from four aspects. First, we identified 15 code review practices for requesting changes to PRs and explored the perception gap between submitters and reviewers in terms of the perceived importance of these practices. Second, we investigated developers' willingness to perform requested changes in terms of revision patience, contribution context, and priority order of change types. Third, we revealed 11 factors causing avoidable revisions from developers' self-reported experience. Finally, we manually examined the reasons for nonacceptance of repeatedly revised PRs and compared the actual reason distribution with developers' mental model.

Our findings have direct implications for PR practitioners: reviewers can follow the identified best practices to promote the continuous revision and improvement of PRs, and reviewers and submitters should pay special attention on the revealed problems causing avoidable revisions and PR nonacceptance for more efficient collaboration in PR revisions. We also proposed suggestions for the design of collaborative environments, *e.g.*, indicating developers' availability and monitoring sentiment in review comments.

As future work, we plan an in-depth analysis of the effect of the number of PR revisions on developers' enthusiasm and contribution quality in subsequent participation in the projects. In addition, we would like to investigate the characteristics of developers who are more willing to perform repeated revisions to PRs and their chances of long-term engagement in a project. Our goal with this research is to provide implications for OSS maintainers and platforms on efficiently identifying, recommending, and attracting potential persistent and long-term contributors. Moreover, it is interesting to further investigate how various factors, including community conventions, project characteristics (*e.g.*, size and age), and demographic attributes (*e.g.*, gender and nationality) and personality traits of reviewers, affect the adoption of the review practices across different projects. Such a broad and comprehensive quantitative analysis would be a useful complement to the current study.

## 11 ACKNOWLEDGMENT

## REFERENCES

[1] Surveys. https://figshare.com/s/7c19599e39664c16eb3f. Accessed: 2021-10-06.

[2] Mark S Ackerman, Juri Dachtera, Volkmar Pipek, and Volker Wulf. Sharing knowledge and expertise: The cscw view of knowledge management. *Computer Supported Cooperative Work*, 22(4-6):531–573, 2013.

[3] Guilherme Avelino, Eleni Constantinou, Marco Tulio Valente, and Alexander Serebrenik. On the abandonment and survival of open source projects: An empirical investigation. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–12. IEEE, 2019.

[4] Alberto Bacchelli and Christian Bird. Expectations, outcomes, and challenges of modern code review. In *2013 35th International Conference on Software Engineering*, pages 712–721. IEEE, 2013.

[5] Andrew Begel and Thomas Zimmermann. Analyze this! 145 questions for data scientists in software engineering. In *Proceedings of the 36th International Conference on Software Engineering*, pages 12–23, 2014.

[6] Moritz Beller, Alberto Bacchelli, Andy Zaidman, and Elmar Juergens. Modern code reviews in open-source projects: Which problems do they fix? In *Proceedings of the 11th working conference on mining software repositories*, pages 202–211, 2014.

[7] Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society*, 57(1):289–300, 1995.

[8] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. What makes a good bug report? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 308–318, 2008.

[9] John Bitchener, Stuart Young, and Denise Cameron. The effect of different types of corrective feedback on esl student writing. *Journal of second language writing*, 14(3):191–205, 2005.

[10] Hudson Borges, Andre Hora, and Marco Tulio Valente. Understanding the factors that impact the popularity of github repositories. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 334–344. IEEE, 2016.

[11] Hudson Borges and Marco Tulio Valente. What's in a github star? understanding repository starring practices in a social coding platform. *Journal of Systems and Software*, 146:112–129, 2018.

[12] Amiangshu Bosu, Jeffrey C Carver, Christian Bird, Jonathan Orbeck, and Christopher Chockley. Process aspects and social dynamics of contemporary code review: Insights from open source development and industrial practice at microsoft. *IEEE Transactions on Software Engineering*, 43(1):56–75, 2016.

[13] Amiangshu Bosu, Michaela Greiler, and Christian Bird. Characteristics of useful code reviews: An empirical study at microsoft. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 146–156. IEEE, 2015.

[14] Silvia Breu, Rahul Premraj, Jonathan Sillito, and Thomas Zimmermann. Information needs in bug reports: improving cooperation between developers and users. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, pages 301–310, 2010.

[15] Caius Brindescu, Iftekhar Ahmed, Rafael Leano, and Anita Sarma. Planning for untangling: Predicting the difficulty of merge conflicts. In *2020 IEEE/ACM 42nd International Conference on Software Engineering*, pages 801–811. IEEE, 2020.

[16] Chris Brown and Chris Parnin. Understanding the impact of github suggested changes on recommendations between developers. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1065–1076, 2020.

[17] Nathan Cassee, Bogdan Vasilescu, and Alexander Serebrenik. The silent helper: the impact of continuous integration on code reviews. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 423–434. IEEE, 2020.

[18] Oscar Chaparro, Jing Lu, Fiorella Zampetti, Laura Moreno, Massimiliano Di Penta, Andrian Marcus, Gabriele Bavota, and Vincent Ng. Detecting missing information in bug descriptions. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 396–407, 2017.

[19] Tommaso Dal Sasso, Andrea Mocci, and Michele Lanza. What makes a satisficing bug report? In *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 164–174. IEEE, 2016.

[20] Nhan Cach Dang, María N Moreno-García, and Fernando De la Prieta. Sentiment analysis based on deep learning: A comparative study. *Electronics*, 9(3):483, 2020.

[21] Giuseppe Destefanis, Marco Ortu, Steve Counsell, Stephen Swift, Michele Marchesi, and Roberto Tonelli. Software development: do good manners matter? *PeerJ Computer Science*, 2:e73, 2016.

[22] Paraminder Dhillon. How to be a good peer reviewer of scientific manuscripts, 2021.

[23] Andrea Di Sorbo, Sebastiano Panichella, Carol V Alexandru, Junji Shimagaki, Corrado A Visaggio, Gerardo Canfora, and Harald C Gall. What would users change in my app? summarizing app reviews for recommending software changes. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 499–510, 2016.

[24] Edson Dias, Paulo Meirelles, Fernando Castor, Igor Steinmacher, Igor Wiese, and Gustavo Pinto. What makes a great maintainer of open source projects? In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 982–994. IEEE, 2021.

[25] Felipe Ebert, Fernando Castor, Nicole Novielli, and Alexander Serebrenik. Confusion in code reviews: Reasons, impacts, and coping strategies. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering*, pages 49–60. IEEE, 2019.

[26] Felipe Ebert, Fernando Castor, Nicole Novielli, and Alexander Serebrenik. An exploratory study on confusion in code reviews. *Empirical Software Engineering*, 26(1):1–48, 2021.

[27] Vasiliki Efstathiou and Diomidis Spinellis. Code review comments: language matters. In *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, pages 69–72, 2018.

[28] Omar Elazhary, Margaret-Anne Storey, Neil Ernst, and Andy Zaidman. Do as i do, not as i say: Do contribution guidelines match the github contribution process? In *2019 IEEE International Conference on Software Maintenance and Evolution*, pages 286–290. IEEE, 2019.

[29] Michael Fagan. Design and code inspections to reduce errors in program development. In *Software pioneers*, pages 575–607. Springer, 2002.

[30] Hongbo Fang, Daniel Klug, Hemank Lamba, James Herbsleb, and Bogdan Vasilescu. Need for tweet: How open source developers talk about their github work on twitter. In *Proceedings of the 17th International Conference on Mining Software Repositories*, pages 322–326, 2020.

[31] Dana R Ferris. Responding to student writing: Teachers' philosophies and practices. *Assessing Writing*, 19:6–23, 2014.

[32] Patricia I Fusch and Lawrence R Ness. Are we there yet? data saturation in qualitative research. *The qualitative report*, 20(9):1408, 2015.

[33] Daviti Gachechiladze, Filippo Lanubile, Nicole Novielli, and Alexander Serebrenik. Anger and its direction in collaborative software development. In *2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track*, pages 11–14. IEEE, 2017.

[34] Ying Gao, Christian Dieter D Schunn, and Qiuying Yu. The alignment of written peer feedback with draft problems and its impact on revision in peer assessment. *Assessment & Evaluation in Higher Education*, 44(2):294–308, 2019.

[35] Marco Gerosa, Igor Wiese, Bianca Trinkenreich, Georg Link, Gregorio Robles, Christoph Treude, Igor Steinmacher, and Anita Sarma. The shifting sands of motivation: Revisiting what drives contributors in open source. *arXiv preprint arXiv:2101.10291*, 2021.

[36] Sarah Gielen, Elien Peeters, Filip Dochy, Patrick Onghena, and Katrien Struyven. Improving the effectiveness of peer feedback for learning. *Learning and instruction*, 20(4):304–315, 2010.

[37] Ketevan Glonti, Isabelle Boutron, David Moher, and Darko Hren. Journal editors' perspectives on the roles and tasks of peer reviewers in biomedical journals: a qualitative study. *BMJ open*, 9(11):e033421, 2019.

[38] Georgios Gousios, Martin Pinzger, and Arie van Deursen. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*, pages 345–355, 2014.

[39] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. Work practices and challenges in pull-based development: the contributor's perspective. In *Proceedings of the 38th International Conference on Software Engineering*, pages 285–296. IEEE, 2016.

[40] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie Van Deursen. Work practices and challenges in pull-based development: The integrator's perspective. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 358–368. IEEE, 2015.

[41] Danielle Guenette. Is feedback pedagogically correct?: Research design issues in studies of feedback on writing. *Journal of second language writing*, 16(1):40–53, 2007.

[42] Philip J Guo, Thomas Zimmermann, Nachiappan Nagappan, and Brendan Murphy. Characterizing and predicting which bugs get fixed: an empirical study of microsoft windows. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 495–504, 2010.

[43] Carl Gutwin, Reagan Penner, and Kevin Schneider. Group awareness in distributed software development. In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, pages 72–81. ACM, 2004.

[44] Samir Haffar, Fateh Bazerbachi, and M Hassan Murad. Peer review bias: a critical review. In *Mayo Clinic Proceedings*, volume 94, pages 670–676. Elsevier, 2019.

[45] Wenjian Huang, Tun Lu, Haiyi Zhu, Guo Li, and Ning Gu. Effectiveness of conflict management strategies in peer review process of online collaboration projects. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, pages 717–728, 2016.

[46] Yu Huang, Denae Ford, and Thomas Zimmermann. Leaving my fingerprints: Motivations and challenges of contributing to oss for social good. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1020–1032. IEEE, 2021.

[47] Bart Huisman, Nadira Saab, Jan Van Driel, and Paul Van Den Broek. Peer feedback on academic writing: undergraduate students' peer feedback role, peer feedback perceptions and essay performance. *Assessment & Evaluation in Higher Education*, 43(6):955–968, 2018.

[48] Giuseppe Iaffaldano, Igor Steinmacher, Fabio Calefato, Marco Gerosa, and Filippo Lanubile. Why do developers take breaks from contributing to oss projects?: a preliminary analysis. In *Proceedings of the 2nd International Workshop on Software Health*, pages 9–16. IEEE Press, 2019.

[49] Juhani Iivari. How to improve the quality of peer reviews? three suggestions for system-level changes. *Communications of the Association for Information Systems*, 38(1):12, 2016.

[50] Rahul N Iyer, S Alex Yun, Meiyappan Nagappan, and Jesse Hoey. Effects of personality traits on pull request acceptance. *IEEE Transactions on Software Engineering*, 2019.

[51] Jing Jiang, Abdillah Mohamed, and Li Zhang. What are the characteristics of reopened pull requests? a case study on open source projects in github. *IEEE Access*, 7:102751–102761, 2019.

[52] Miguel Jiménez, Mario Piattini, and Aurora Vizcaíno. Challenges and improvements in distributed software development: A systematic review. *Advances in Software Engineering*, 2009, 2009.

[53] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. The promises and perils of mining github. In *Proceedings of the 11th working conference on mining software repositories*, pages 92–101, 2014.

[54] David Kavaler, Premkumar Devanbu, and Vladimir Filkov. Whom are you going to call? determinants of @-mentions in github discussions. *Empirical Software Engineering*, 24(6):3904–3932, 2019.

[55] Hammad Khalid, Emad Shihab, Meiyappan Nagappan, and Ahmed E Hassan. What do mobile app users complain about? *IEEE software*, 32(3):70–77, 2014.

[56] Oleksii Kononenko, Olga Baysal, and Michael W Godfrey. Code review quality: how developers see it. In *Proceedings of the 38th International Conference on Software Engineering*, pages 1028–1038, 2016.

[57] Chinmay E Kulkarni, Michael S Bernstein, and Scott R Klemmer. Peerstudio: rapid peer feedback emphasizes revision and improves performance. In *Proceedings of the second (2015) ACM conference on learning@ scale*, pages 75–84, 2015.

[58] Karim R Lakhani and Robert G Wolf. Why hackers do what they do: Understanding motivation and effort in free/open source software projects. 2003.

[59] Eero I Laukkanen and Mika V Mantyla. Survey reproduction of defect reporting in industrial software development. In *2011 International Symposium on Empirical Software Engineering and Measurement*, pages 197–206. IEEE, 2011.

[60] Amanda Lee, Jeffrey C Carver, and Amiangshu Bosu. Understanding the impressions, motivations, and barriers of one time code contributors to floss projects: a survey. In *Proceedings of the 39th International Conference on Software Engineering*, pages 187–197. IEEE Press, 2017.

[61] Icy Lee. Understanding teachers' written feedback practices in hong kong secondary classrooms. *Journal of second language writing*, 17(2):69–85, 2008.

[62] Zhixing Li, Yue Yu, Tao Wang, Gang Yin, Shanshan Li, and Huaimin Wang. Are you still working on this? an empirical study on pull request abandonment. *IEEE Transactions on Software Engineering*, pages 1–1, 2021.

[63] Laura MacLeod, Michaela Greiler, Margaret-Anne Storey, Christian Bird, and Jacek Czerwonka. Code reviewing in the trenches: Challenges and best practices. *IEEE Software*, 35(4):34–42, 2017.

[64] Mika V Mäntylä and Casper Lassenius. What types of defects are really discovered in code reviews? *IEEE Transactions on Software Engineering*, 35(3):430–448, 2008.

[65] Jennifer Marlow, Laura Dabbish, and Jim 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*, pages 117–128. ACM, 2013.

[66] Hedy McGarrell and Jeff Verbeem. Motivating revision of drafts through formative feedback. *ELT journal*, 61(3):228–236, 2007.

[67] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 192–201, 2014.

[68] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering*, 21(5):2146–2189, 2016.

[69] Courtney Miller, David Gray Widder, Christian Kästner, and Bogdan Vasilescu. Why do people give up flossing? a study of contributor disengagement in open source. In *Proceedings of the 2019 International Conference on Open Source Systems*, pages 116–129. Springer, 2019.

[70] Hui-Tzu Min. The effects of trained peer review on efl students' revision types and writing quality. *Journal of second language writing*, 15(2):118–141, 2006.

[71] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. Curating github for engineered software projects. *Empirical Software Engineering*, 22(6):3219–3253, 2017.

[72] Nicole Novielli, Fabio Calefato, and Filippo Lanubile. The challenges of sentiment detection in the social programmer ecosystem. In *Proceedings of the 7th International Workshop on Social Software Engineering*, pages 33–40, 2015.

[73] Nicole Novielli, Daniela Girardi, and Filippo Lanubile. A benchmark study on sentiment analysis for software engineering research. In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories*, pages 364–375. IEEE, 2018.

[74] Jeungmin Oh, Daehoon Kim, Uichin Lee, Jae-Gil Lee, and Junehwa Song. Facilitating developer-user interactions with mobile app review digests. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, pages 1809–1814. 2013.

[75] Dennis Pagano and Walid Maalej. User feedback in the appstore: An empirical study. In *2013 21st IEEE international requirements engineering conference (RE)*, pages 125–134. IEEE, 2013.

[76] Sebastiano Panichella and Nik Zaugg. An empirical investigation of relevant changes and automation needs in modern code review. *Empirical Software Engineering*, 25(6):4833–4872, 2020.

[77] Luca Pascarella, Davide Spadini, Fabio Palomba, Magiel Bruntink, and Alberto Bacchelli. Information needs in contemporary code review. *Proceedings of the ACM on Human-Computer Interaction*, 2(CSCW):1–27, 2018.

[78] Trena M Paulus. The effect of peer and teacher feedback on student writing. *Journal of second language writing*, 8(3):265–289, 1999.

[79] Raphael Pham, Leif Singer, Olga Liskin, Fernando Figueira Filho, and Kurt Schneider. Creating a shared understanding of testing culture on a social coding site. In *Proceedings of the 2013 International Conference on Software Engineering*, 2013.

[80] Gustavo Pinto, Igor Steinmacher, and Marco Aurélio Gerosa. More common than you think: An in-depth study of casual contributors. In *Proceedings of the 23rd International Conference on Software Analysis, Evolution, and Reengineering*, volume 1, pages 112–123. IEEE, 2016.

[81] Huilian Sophie Qiu, Yucen Lily Li, Susmita Padala, Anita Sarma, and Bogdan Vasilescu. The signals that potential contributors look for when choosing open-source projects. *Proceedings of the ACM on Human-Computer Interaction*, 3(CSCW):1–29, 2019.

[82] Mohammad Masudur Rahman, Chanchal K Roy, and Raula G Kula. Predicting usefulness of code review comments using textual features and developer experience. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories*, pages 215–226. IEEE, 2017.

[83] Sarah Rastkar, Gail C Murphy, and Gabriel Murray. Automatic summarization of bug reports. *IEEE Transactions on Software Engineering*, 40(4):366–380, 2014.

[84] David B Resnik and Susan A Elmore. Ensuring the quality, fairness, and integrity of journal peer review: A possible role of editors. *Science and Engineering Ethics*, 22(1):169–188, 2016.

[85] Dirk Riehle, Philipp Riemer, Carsten Kolassa, and Michael Schmidt. Paid vs. volunteer work in open source. In *2014 47th Hawaii International Conference on System Sciences*, pages 3286–3295. IEEE, 2014.

[86] Peter C Rigby and Christian Bird. Convergent contemporary software peer review practices. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pages 202–212, 2013.

[87] Julia Rubin and Martin Rinard. The challenges of staying together while moving fast: An exploratory study. In *2016 IEEE/ACM 38th International Conference on Software Engineering*, pages 982–993. IEEE, 2016.

[88] Andreas Schilling, Sven Laumer, and Tim Weitzel. Together but apart: How spatial, temporal and cultural distances affect floss developers' project retention. In *Proceedings of the 2013 annual conference on Computers and people research*, pages 167–172, 2013.

[89] Natalie Shoham and Alexandra Pitman. Open versus blind peer review: is anonymity better than transparency? *BJPsych Advances*, 27(4):247–254, 2021.

[90] Simon Sinek. *Start with why: How great leaders inspire everyone to take action.* Penguin, 2009.

[91] Leif Singer, Fernando Figueira Filho, Brendan Cleary, Christoph Treude, Margaret-Anne Storey, and Kurt Schneider. Mutual assessment in the social programmer ecosystem: An empirical investigation of developer profile aggregators. In *Proceedings of the 2013 conference on Computer supported cooperative work*, pages 103–116, 2013.

[92] Devarshi Singh, Varun Ramachandra Sekar, Kathryn T Stolee, and Brittany Johnson. Evaluating how static analysis tools can reduce code review effort. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 101–105. IEEE, 2017.

[93] Devlin V Smith, Laura B Stokes, Kayleigh Marx, and Samuel L Aitken. Navigating manuscript assessment: The new practitioner's guide to primary literature peer review. *Journal of oncology pharmacy practice*, 25(1):94–100, 2019.

[94] Igor Steinmacher, Tayana Conte, Marco Aurélio Gerosa, and David 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. ACM, 2015.

[95] Igor Steinmacher, Gustavo Pinto, Igor Scaliante Wiese, and Marco Aurélio Gerosa. Almost there: A study on quasi-contributors in open-source software projects. In *2018 IEEE/ACM 40th International Conference on Software Engineering*, pages 256–266. IEEE, 2018.

[96] Igor Steinmacher, Igor Wiese, Ana Paula Chaves, and Marco Aurélio Gerosa. Why do newcomers abandon open source software projects? In *Proceedings of the 6th International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 25–32. IEEE, 2013.

[97] Margaret-Anne Storey, Leif Singer, Brendan Cleary, Fernando Figueira Filho, and Alexey Zagalsky. The (r) evolution of social media in software engineering. In *Future of Software Engineering Proceedings*, pages 100–116. 2014.

[98] Mohammadali Tavakoli, Liping Zhao, Atefeh Heydari, and Goran Nenadić. Extracting useful software development information from mobile application reviews: A survey of intelligent mining techniques and tools. *Expert Systems with Applications*, 113:186–199, 2018.

[99] Josh Terrell, Andrew Kofink, Justin Middleton, Clarissa Rainear, Emerson Murphy-Hill, Chris Parnin, and Jon Stallings. Gender differences and bias in open source: Pull request acceptance of women versus men. *PeerJ Computer Science*, 3:e111, 2017.

[100] Parastou Tourani, Bram Adams, and Alexander Serebrenik. Code of conduct in open source projects. In *2017 IEEE 24th international conference on software analysis, evolution and reengineering*, pages 24–33. IEEE, 2017.

[101] Jason Tsay, Laura Dabbish, and James Herbsleb. Influence of social and technical factors for evaluating contribution in github. In *Proceedings of the 36th International Conference on Software Engineering*, pages 356–366. ACM, 2014.

[102] Jason Tsay, Laura Dabbish, and James 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*, pages 144–154, 2014.

[103] Bogdan Vasilescu, Kelly Blincoe, Qi Xuan, Casey Casalnuovo, Daniela Damian, Premkumar Devanbu, and Vladimir Filkov. The sky is not the limit: multitasking across github projects. In *Proceedings of the 38th International Conference on Software Engineering*, pages 994–1005, 2016.

[104] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. Quality and productivity outcomes relating to continuous integration in github. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 805–816, 2015.

[105] Ravichandran Vengadasamy. Responding to student writing: Motivate, not criticise. *GEMA Online® Journal of Language Studies*, 2, 2002.

[106] Mark Ware. *Peer review: benefits, perceptions and alternatives*. Citeseer, 2008.

[107] Mairieli Wessel, Bruno Mendes de Souza, Igor Steinmacher, Igor S. Wiese, Ivanilton Polato, Ana Paula Chaves, and Marco A. Gerosa. The power of bots: Characterizing and understanding bots in oss projects. *Proceedings of the 2018 ACM on Human-Computer Interaction*, 2(CSCW), November 2018.

[108] Mairieli Wessel, Alexander Serebrenik, Igor Wiese, Igor Steinmacher, and Marco A Gerosa. What to expect from code review bots on github? a survey with oss maintainers. In *Proceedings of the 34th Brazilian Symposium on Software Engineering*, pages 457–462, 2020.

[109] Yong Wu and Christian D Schunn. From feedback to revisions: Effects of feedback features and perceptions. *Contemporary Educational Psychology*, 60:101826, 2020.

[110] Miao Yang, Richard Badger, and Zhen Yu. A comparative study of peer and teacher feedback in a chinese efl writing class. *Journal of second language writing*, 15(3):179–200, 2006.

[111] Yunwen Ye and Kouichi Kishida. Toward an understanding of the motivation open source software developers. In *Proceedings of the 25th International Conference on Software Engineerin*, pages 419–429. IEEE Computer Society, 2003.

[112] Yue Yu, Huaimin Wang, Gang Yin, and Tao Wang. Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment? *Information and Software Technology*, 74:204–218, 2016.

[113] Yue Yu, Gang Yin, Tao Wang, Cheng Yang, and Huaimin Wang. Determinants of pull-based development in the context of continuous integration. *Science China Information Sciences*, 59(8):080104, 2016.

[114] Fiorella Zampetti, Gabriele Bavota, Gerardo Canfora, and Massimiliano Di Penta. A study on the interplay between pull request review and continuous integration builds. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering*, pages 38–48. IEEE, 2019.

[115] Motahareh Bahrami Zanjani, Huzefa Kagdi, and Christian Bird. Automatically recommending peer reviewers in modern code review. *IEEE Transactions on Software Engineering*, 42(6):530–543, 2016.

[116] Shurui Zhou, Bogdan Vasilescu, and Christian Kästner. What the fork: a study of inefficient and efficient forking practices in social coding. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 350–361, 2019.

[117] Thomas Zimmermann. Card-sorting: From text to themes. In *Perspectives on Data Science for Software Engineering*, pages 137–141. Elsevier, 2016.

[118] Weiqin Zou, Jifeng Xuan, Xiaoyuan Xie, Zhenyu Chen, and Baowen Xu. How does code style inconsistency affect pull request integration? an exploratory study on 117 github projects. *Empirical Software Engineering*, 24(6):3871–3903, 2019.