An Empirical Study of Overlooked Code Review Comments in OSS Projects

Yike Li, Yuxia Zhang, Qunhong Zeng, Lin Shi, Xin Tan, Tao Wang, Yanjie Jiang, Hui Liu

Abstract—Open source software (OSS) development widely adopts modern code review to identify issues and guarantee code quality. As reported repeatedly, maintainers are under heavy workloads when reviewing code changes. Meanwhile, we notice that some code reviews were overlooked by the authors of the code changes, i.e., neither causing code modification nor being replied to. These code reviews, if requiring responses but not receiving any, might represent a significant inefficiency, risk of overlooking critical issues, and problematic social exchange. Moreover, leaving code reviews publicly unanswered may cause a negative impression on both the corresponding OSS contributors and the OSS projects. Existing literature on code review mainly focuses on the usefulness of code reviews, reviewer recommendations, factors affecting PR acceptance, and review comment generation; the nature of overlooked reviews has not been explored. To this end, we focus on a widely-used modern code review mechanism, i.e., reviewing Pull Request (PR) code before merge, and conduct the first empirical study on 80 Java OSS projects to explore the prevalence, characteristics, rationales, and possible impact of the overlooked code reviews. We find that approximately 7.5% of PRs have at least one review comment being ignored. We further show that pull requests containing no-response comments are significantly associated with longer review lifecycles and lower acceptance rates, indicating measurable negative outcomes beyond their modest prevalence. Then, we categorize these no-response comments through thematic analysis and find two main categories with seven subcategories: Review inquiry and PR management. We also extract four subcategories in Review inquiry, e.g., Give suggestions about code implementation, Point out implementation issues, and Additional task requests. PR management consists of three subcategories, i.e., PR status checks, PR merge conflict notifications, and Reject PR with uncertain reasons. To better understand the existence of no-response comments, we surveyed developers and received 45 responses. We found that the reasons for the existence of no-response comments are diverse, such as prolonged review times and a lack of consensus on opinions. Developers also hold the consensus that ignored reviews will have negative effects on software projects. These findings emphasize the need for attention from both academia and industry to the responses to review comments and optimization of the reminder mechanism.

Index Terms—Modern code review, pull requests, overlooked review comments

1 Introduction

Peer Code Review is the act of checking source code written by a collaborator to judge whether it is good enough to be merged into the main codebase [1], [2]. It is an important software development activity that can find defects, share knowledge, and maintain code quality [3]. Early peer code review has a formalized and structured process, which requires line-by-line code inspections [4]. Later, a more lightweight code review practice gained wide adoption from both industrial and open source software (OSS) projects. This code review practice is named "Modern code review" [5], [6], which is tool-based, informal, and asynchronous. With the increasing application of online collaborative platforms, GitHub in particular, millions of OSS projects conduct modern code reviews openly and transparently. Therefore, these shared code reviews can not only assure the quality of the corresponding code changes but also provide a wide spectrum of benefits to software teams, such as knowledge transfer, team awareness, and improved solutions to problems [5].

However, it is time-consuming to manually review other collaborators' code. As reported, developers spend approximately 10-15% of their time in code reviews [3]. Also, researchers have found that thousands of code reviews need to be conducted in some mature industrial projects per month [7]. In the context of open source projects, the impact of requiring substantial time might be worse, since developers can only do code reviews in their spare time, and failure to receive a response for a long time can cause newcomers to leave the open source community [8]. Therefore, numerous studies have explored ways to improve or optimize code review efficiency. One stream of research has focused on exploring and addressing the challenges of modern code review [5]. Zhou et al. [9] and Tan et al. [10] conveyed that the maintainers in the Linux kernel community are under a great burden of code review. Bosu et al. [11] found that approximately 34.5% of code review comments are not useful by conducting an empirical study at Microsoft. After that, several studies focus on automatically classifying the usefulness of comments [11]–[14]. The other stream of research is dedicated to automating code review activities by leveraging deep learning models and large language models [15]–[17].

- *Yike Li, Yuxia Zhang, Qunhong Zeng, and Hui Liu are with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China. Email: {liyike1, qunhongzeng, liuhui08, yuxiazh}@bit.edu.cn(Corresponding author: Yuxia Zhang.)*
- *Lin Shi is with the School of Software, Beihang University, China. E-mail: shilin@buaa.edu.cn*
- *Xin Tan is with the School of Computer Science, Beihang University, China. E-mail: xintan@buaa.edu.cn*
- *Yanjie Jiang is with the School of Computer Science, Peking University, Beijing, China. E-mail: yanjiejiang@pku.edu.cn.*
- *Tao Wang is with the School of Computer Science, National University of Defense Technology, Hunan, China. E-mail: taowang2005@nudt.edu.cn.*

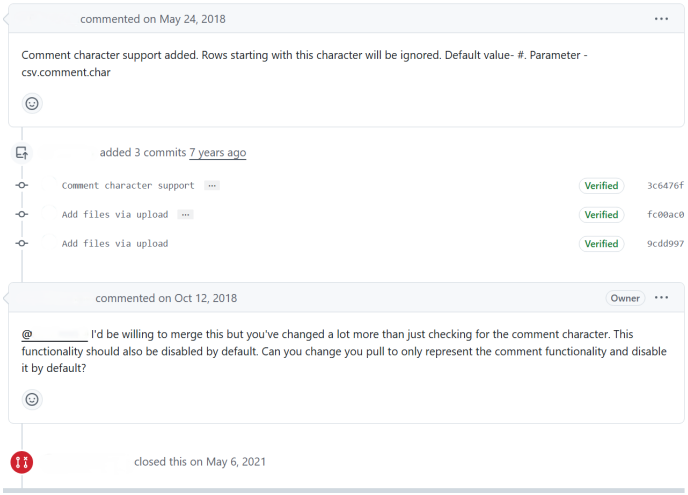

Fig. 1: An example of a no-response comment.

Although both academia and industry are dedicated to improving the efficiency of code review, we find that there are review comments without getting any feedback, i.e., neither causing code changes nor being replied to, though a response is required (we noted as ***"no-response comments"***). The existence of this kind of review comment is not only a waste of the corresponding reviewers' time but may also threaten the sustainability of the project. Figure 1 shows an example[1] of a pull request (PR) where a feature to support comment character handling in CSV files was added. The reviewer suggests that the PR introduces changes beyond the intended scope, requesting that the functionality be disabled by default and limited to the comment character feature. However, this comment got no feedback, leading to the close of this PR without resolution. The significance of investigating no-response comments is also reflected in long-established community guidelines within large-scale open-source ecosystems. For instance, the Linux community's official development process explicitly reminds contributors not to ignore review comments[2], underscoring that responsiveness to feedback is a fundamental expectation of collaborative software development. To the best of our knowledge, no prior work has analyzed the existence and characteristics of these no-response comments. Hence, we conduct the first empirical study of the no-response comments in modern code reviews to bridge the knowledge gap.

The goal of our study can be divided into three research questions. First, we are eager to know: How prevalent are review comments being overlooked during the code review process (RQ1)? This phenomenon highlights potential communication gaps and inefficiencies in the review process. By quantifying this phenomenon, we can understand how serious it is and examine whether having no-response comments is associated with observable negative review outcomes. To go a step further, is the content of the comments causing it to receive no response? Do these no-response comments share some common characteristics? Therefore, we ask: What types of review comments have been ignored (RQ2)? Based on the findings of RQ2, we are interested to know: How do developers perceive no-response comments (RQ3)? Specifically, we explore whether they have ever experienced no-response comments, how common, and why. Understanding the rationale of the no-response comments can help OSS communities and academia explore ways to improve the efficiency and quality of code review.

We conducted a mixed-method study to answer the three research questions. Firstly, we stratified-sampled 80 Java projects hosted on GitHub based on their PR numbers. For each project, we randomly selected 10 PRs. Some small repositories have less than 10 PRs that have at least one comment from reviewers. As a result, we annotated 760 PRs that have 2,104 review comments in total, by determining whether the comments had received a response. We found that 57 (7.5%) PRs have no-response comments. The number of no-response comments in the 57 PRs ranges from one to twelve, with 92 in total. Although no-response comments constitute only a modest fraction, this rate can represent thousands of unanswered reviewer comments across more than 232,000 PRs in our dataset, and the statistical results indicate that PRs with no-response comments are significantly correlated with being rejected and longer handling time. Then we applied thematic analysis to the 92 no-response comments to characterize their content and extract categories. Specifically, we found two categories of no-response comments: Review inquiry and PR management. Review inquiry consists of four subcategories, i.e., give suggestions about code implementation, point out implementation issues, ask questions regarding code implementation, and additional task requests. PR management consists of three subcategories, i.e., PR status checks, PR merge conflict notifications, and reject PR with uncertain reasons. Finally, we surveyed developers' perspectives on giving no responses to review comments. After analyzing the 45 responses received, we found a variety of reasons, e.g., waiting too long, lack of time, and logical and functional difficulties. They all agreed that disregarding these comments could negatively impact OSS projects. Our results show that ignoring review comments is a socio-technical phenomenon that deserves attention.

This work makes the following contributions:

- We conducted the first empirical study on no-response comments in modern code review by quantifying their prevalence and relationship with review outcomes.
- We categorized the patterns of no-response comments based on their characteristics, bridging the knowledge gap toward understanding no-response code reviews.
- We provided a typology of the reasons why these review comments tend to be neglected, which can assist in improving the efficiency of future code reviews.

In the remainder of this paper, we review related work in Sec. 2, outline our multi-method research approach in Sec. 3, and present the results of our study in Sec. 4. We discuss the implications for research and practice in Sec. 5. We present threats to the validity of our reported findings in Sec. 6 and conclude the paper in Sec. 7.

2 Related Work

In this section, we review prior research related to modern code review, with a focus on four critical aspects that

1. https://github.com/jcustenborder/kafka-connect-spooldir/pull/55
2. https://www.kernel.org/doc/html/v4.16/process/howto.html

inform our study: review usefulness, improvement of code review quality, factors affecting PR acceptance and automatic generation of code review. Overlooked review comments miss the opportunities to play their roles and may be explored when studying review usefulness and quality. On the other hand, AI for SE is widely explored, and overlooked review comments may share some common characteristics, which should be considered when generating code reviews.

2.1 Review Usefulness

Numerous studies have concentrated on modern code review. The first stream focuses on examining their impact on code quality. For instance, Beller et al. explored the types of issues addressed in modern code review and found that 10-35% of review recommendations did not result in code changes [18]. They also found that tasks with more modified files and higher code churn experienced more changes compared to bug-fixing tasks, and reviewer identity information did not affect the number of code changes [18]. McIntosh et al. investigated the relationship between post-release defects and factors related to code review, such as code review coverage, participation, and reviewer expertise. By quantitatively examining three large software systems utilizing contemporary code review tools, they concluded that code review coverage, participation, and expertise were significantly associated with software quality [19]. Bosu et al. [11] explore the characteristics of useful code review comments by interviewing professional developers and analyzing 1.5 million comments from five projects at Microsoft, and they identified numerous factors affecting the usefulness of review comments, including the reviewer's experience, change scale, file type, etc. After that, a series of studies were dedicated to classifying useful and non-useful comments by leveraging various characteristics of comments and different algorithms [12]–[14]. Although no-response comments may be treated as 'useless' and neglected by the author of code changes, prior literature has conducted little investigation about this.

2.2 Improvement of Code Review Quality

Since the positive impact of code review has been widely recognized, researchers have started to investigate how to improve the efficiency of code review. Spadini et al. posited that comments made on a specific code change could influence other reviewers' judgments of the code, thereby affecting the entire project's code quality [20]. Through an empirical study examining code review quality for a large open-source project, Kononenko et al. emphasized the significance of individuals and participation during the code review process [21]. In a subsequent study, they explored how developers perceive code review quality [22]. Their findings suggested that review quality is primarily associated with the thoroughness of feedback, reviewers' familiarity with the code, and the perceived quality of the code itself. On the other hand, researchers try to improve code review efficiency by recommending experienced reviewers [23]–[25]. For example, Thongtanunam et al. proposed a reviewer recommendation algorithm, which is based on file path similarity [23]. Similarly, Zanjani et al. also presented an approach to automatically recommend peer reviewers in modern code review. They demonstrated the effectiveness of the method through multi-perspective empirical evaluations on both industrial platforms and open source systems. [24]. In Guoping Rong et al.'s work [25], they adopt the hypergraph technique to model the high-order relationship (i.e., one PR with multiple reviewers herein) and develop a new recommender, namely HGRec. HGRec is more likely to recommend a diversity of reviewers, which can help relieve the core reviewers' workload congestion issue.

2.3 Factors Affecting PR Acceptance

Researchers also explored which factors can affect the acceptance of PRs. Tsay et al. [26] conducted an empirical study on GitHub PR acceptance by modeling both technical features (e.g., size of code change and presence of tests) and social signals (e.g., prior interactions, social network distance between contributor and integrator), showing that both classes of factors contribute to the decision and that extensive discussion (many comments) may negatively correlate with acceptance under some circumstances. Dey & Mockus [27] extended this line of work in the NPM ecosystem: they proposed a set of technical and social hypotheses, defined 17 predictors (including ecosystem-level measures), and showed that most are significant, sometimes in a nonlinear fashion, in predicting PR acceptance. More recently, Zhang et al. [28] collected 3,347,937 PRs from 11,230 GitHub projects with 95 candidate features, and built mixed-effects logistic regression models; they identify a compact set of highly predictive features and find that the factor "whether the integrator is the same as the submitter" is among the strongest predictors. In addition, Krutauz et al. [29] replicated McIntosh et al.'s analysis on new projects and employed Bayesian network modeling, and found that code review predictors often do not exert direct effects on post-release defects but may act through intermediate variables or be unstable under variable selection. Our work differs from and extends these prior studies: while those works focus primarily on active review participation (number of comments, identity of reviewers, social ties, etc.), we explore no-response comments (i.e., overlooked reviewer comments) themselves; having such comments may also affect PR acceptance.

To the best of our knowledge, we conducted the first empirical study with a focus on these no-response comments. Our findings can complement the prior understanding of how to conduct code review efficiently and may assist in review generation studies to prepare datasets.

3 Method

In this section, we introduce how we collect and preprocess data, annotate and characterize no-response comments, and explore the reasons for these no-response comments. Figure 2 shows a method overview of our study. In data collection, we performed stratified sampling on Java projects hosted on GitHub based on the number of pull requests (PRs), with 20 projects in each layer and four layers, leading to a total of 80 projects. We then randomly selected 10 PRs from each project as the dataset for annotation. Next, we manually annotated the review comments within these PRs to determine whether they were "no-response comments."

We then conducted a thematic analysis of the no-response comments to identify different types. Finally, we surveyed developers to investigate the causes and impacts of no-response comments, among other factors.

3.1 Data Collection and Preprocessing

Previous work [30] revealed that programming languages can impact software development differently. In this work, we focus on Java, one of the most widely used programming languages in both industry and academia [31], [32], to conduct this empirical study of no-response reviews. This choice ensures extensive code repositories and rich project data, providing ample sources for analysis. Besides, Java has been the focus of numerous prior empirical studies, enabling comparison and validation of our findings.

Pull request is the main mechanism for conducting code reviews in most projects hosted on GitHub. When developers prepare a code change, they will start a PR asking core developers to review their local branch with the code change. Then reviewers give comments (noted as PR comments) on the change and the PR author will improve the change accordingly or respond to the code review comments. Normally, the PR will be simply closed or with the code change be merged into the project's central branches. In this study, we select Pull request as the main mechanism to explore no-response code reviews. For convenience, we name the developer who starts a PR as "PR author" and the developers who submitted comments in the PR as "PR reviewers". We only studied the comments provided by PR reviewers in the PRs.

To comprehensively investigate the phenomenon of no-response comments in open source communities, we perform a stratified sampling [33] of Java projects based on their number of Pull Requests. Specifically, we search all the Java projects on GitHub by the service of SEART GitHub Search Engine [34], which allows researchers to sample repositories by using several combinations of selection criteria. Our selection criteria are as follows: (1) set 'Language' as Java; (2) set 'Minimum Number of PRs' as 10 which is intuitively used to filter out toy projects or projects that do not rely on the GitHub platform to conduct code review; and (3) set 'Exclude Forks'. We empirically require each selected repository to have at least 10 pull requests, in case selecting projects that do not use Pull Request for code review.

After collecting the eligible Java repositories, we rank them based on the number of PRs and divide them into four equal-sized groups based on the first, second, and third quartiles. From each group, we randomly sample 20 repositories. This stratification enables us to explore whether no-response review comments occur across projects of varying sizes and maturity levels, as indicated by the number of PRs. Thus, we obtained 80 Java projects as study cases. Table 1 shows a statistical summary of the 80 projects.

To collect the review data of the 80 selected projects, we apply GitHub REST API [35], which can collect data of pull requests, including review identifier, author, reviewers, comments, initial and addressed code changes, and time of each event, etc. Since ignored review comments can happen at any stage once a PR has been submitted, we select PRs in all states, i.e., be merged (i.e., the changes are merged into the project code base), closed (i.e., code changes failed to be merged), and open (the pull requests are still in progress). Meanwhile, we ensure that each pull request contains at least one comment to facilitate the exploration of the phenomenon of no-response comments. We obtained all the 232,580 pull request links of these 80 Java projects through the GitHub REST API, and then randomly selected 10 pull requests from each project, resulting in a total of 800 pull requests (with an error margin of 4.5 percent and a confidence level of 99 percent). Among the 800 PRs, the number of comments ranges from 1 to 48, with a median of 2.

3.2 Characterizing No-response Comments

It is hard to automatically determine whether a comment has received any feedback because (1) one updated code change may address multiple comments from different reviewers; (2) comments and replies are usually interspersed with each other. To accurately evaluate the prevalence of no-response comments, we conduct a manual annotation process. Then, for the identified no-response comments, we apply a classical qualitative analysis method, i.e., thematic analysis [36], to categorize the patterns of no-response comments based on their characteristics.

3.2.1 Data Annotation

For a certain review comment, the first two authors determine whether it has received any feedback, including the following steps:

(1) Thoroughly read and understand all the code changes, reviewers' comments, and author's replies in the pull request under annotation;
(2) Analyze each reviewer's comment to determine whether it needs a response;
(3) Check the content of the following updated code changes after the comment to explore whether the issue(s) mentioned in the comment has been addressed. For PRs with a status of closed or merged, we set the period from the point of creating a comment to the end of the PR. For open PRs, we empirically set a three-month window from the time of creating a comment to do the check. This threshold follows prior empirical studies on PR lifecycles showing that the vast majority of review activity happens within the first few weeks [37];
(4) Analyze whether the following replies from both the authors and other reviewers responded to the comment under annotation.
(5) If the answer to step (2) is 'yes' and both the answers to steps (3) and (4) are 'no', we label the comment as 'no-response'.

To ensure clarity and consistency in the coding process, we have developed a codebook for identifying no-response comments, which can be found in the appendix [38].

In the 760 pull requests, 2,104 review comments were analyzed, among which 2,044 comments were assigned the same labels. Cohen's kappa coefficient of agreement [39] between the two annotators was 0.94, indicating a high level of consistency. For the 60 comments with different labels, we held a series of meetings to resolve the conflicts, and an agreement was reached. After annotation, we found 92 comments from 57 PRs that have received no response,

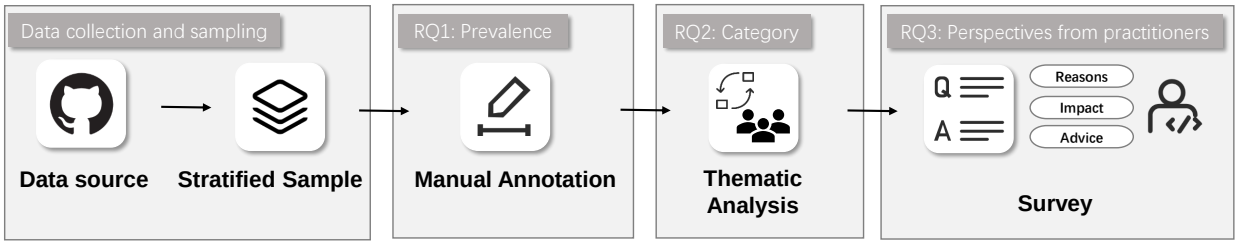

Fig. 2: Method overview of our study.

Table 1: Statistics of the four groups of projects classified by PR number quartiles.

Groups	PR Num Range	#Projects	#Contributors (*mean/median*)	#Star (*mean/median*)	Example
G1	10 ∼ 21	6,377	3 ∼ 981 (*8.7/6.0*)	10 ∼ 10,444 (*231.2/43.0*)	tltv/gantt
G2	22 ∼ 49	5,879	3 ∼ 861 (*12.0/9.0*)	10 ∼ 32,411 (*332.6/53.0*)	goldengnu/jeveassets
G3	50 ∼ 150	6,034	3 ∼ 732 (*17.9/13.0*)	10 ∼ 73,540 (*484.2/59.0*)	ari4java/ari4java
G4	151 ∼ 66,312	6,083	3 ∼ 485 (*48.7/27.0*)	10 ∼ 145,397 (*1,140.8/105.0*)	apache/helix

Table 2: Thematic analysis process for classifying no-response comments.

Text	Codes	Sub-Theme	Theme
How about LenientContentDispositionFieldTest ? We should avoid divergences between both of them . . .	"Test consistency", "Avoid divergence in tests"	Ensure functionality consistency	Give Suggestions about Code Implementation
Add usage for *add usage for ClassWithDottedProperties* , otherwise LGTM.	"Add usage for unimplemented functionality"		
Instead of doing this, it may be better to have a subclass for the case where there are two values. Also, reset would have to reset secondValue too.	"Refactor with subclass", "Ensure consistency in reset behavior"	Optimize code design and implementation	
We don't need to instantiate SimpleMetadataReaderFactory every time this method is called.	"Optimize object instantiation"		
The APK level is set to 24 (Android 7), so this check isn't necessary; it will always pass.	"Remove unnecessary check for fixed APK level"		

accounting for 4.4%. In terms of pull requests, 7.5% of PRs contain no-response comments. Although the percentage of these no-response comments is not that substantial, we deem its cumulative effect on the entire OSS ecosystem is still considerable, because it indicates a huge waste of reviewers' efforts and knowledge and may also hide serious issues.

3.2.2 Thematic Analysis

We apply thematic analysis [36] to extract the characteristics of these no-response comments and summarize categories. Thematic analysis [36] is a method for analyzing semantic data (for example, text, as in our study) that involves reading through a set of data and looking for patterns in the meaning of the data to find themes. Specifically, the first and third authors conducted the thematic analysis as follows:

(1) Carefully read all the context of the no-response comments to understand what kind of issues the reviewers pointed out and mark the key phrases. During this step, we find that some comments require feedback, i.e., asking the author of the code change to fix a specific issue. While some comments are more like closing remarks and have little need for feedback. ***Our study focuses on exploring the review comments that truly need a response; thus, we do not consider these closing remarks.***
(2) Reread these comments and pay more attention to the marked key phrases to generate the initial code.
(3) Systematically analyze the initial code and aggregate code indicating the same issues. In this step, we give initial themes to the aggregated code.
(4) Review the initial themes, merge the ones having similar meanings, and determine the "is a" relationship between themes.
(5) Define the final set of themes for all the no-response comments.

To reduce any manual bias, steps (1) to (4) described above were performed independently by the first two authors of this paper. We also held a sequence of meetings to resolve conflicts and assign the final themes (step 5). Table 2 shows examples of how we conduct thematic analysis.

3.3 Survey for Reasons for No-response Reviews

We initially surveyed the PR authors of the identified no-response PR comments to explore the reasons. However, after two months of waiting and a one-time reminder, we did not receive any response. The reason might be that it has been too long, and they have already left the OSS projects or cannot recall what happened in the corresponding PRs. Also, they may feel uncomfortable when asked these kinds of questions. Thus, we turn to OSS developers to solicit their practical opinions on ignoring code reviews generally.

3.3.1 Survey Design

At the beginning of the survey, we followed the instructions [40] regarding survey design and asked two demo-

Table 3: Survey Questions

Category	Question	Answer Format
Demographic	How long have you been Involved in OSS?	Single selection options: Less than 3 months; 3 months to 1 year; 1 year to 4 years; 4 years to 10 years; and More than 10 years.
	To which gender identity do you most identify?	Single selection options: Male; Female; Prefer not to say; and Not listed.
PR reviewers	As a PR reviewer, have you ever encountered a situation where your comments did not receive a response while reviewing pull requests?	Single selection options: Yes; No; or Can't remember.
	In the past year, how many times have you encountered such situations where your comments did not receive a response?	Single selection options: Less than 4 times; 4 to 6 times; 7 to 9 times; 10 to 12 times; or More than 12 times.
	As a PR reviewer, what do you think could be the reasons for PR authors not responding to your comments?	Multiple selection options: The time waiting for review is too long; The comments are not clear enough and difficult to understand; Lack of time; Didn't agree with the opinion given by the reviewer; Logical and functional difficulties; Low importance; and Other (free text).
PR author	As a PR author, have you ever chosen not to respond to a reviewer's comment in your submitted PRs?	Single selection options: Yes; No; or Can't remember.
	In the past year, how many times have you chosen not to respond to comments from reviewers in your submitted PRs?	Single selection options: Less than 4 times; 4 to 6 times; 7 to 9 times; 10 to 12 times; or More than 12 times.
	As a PR author, what reasons have caused you to not respond to some comments provided by reviewers in your submitted PRs?	Multiple selection options: The time waiting for review is too long; The comments are not clear enough and difficult to understand; Lack of time; Didn't agree with the opinion given by the reviewer; Logical and functional difficulties; Low importance; and Other (free text).
Open Perspectives and Suggestions	How do you perceive the situation when PR authors do not respond to reviews? What impact do you think it may have on the project?	Free text
	To reduce the incidence of such problems, please share your valuable suggestions.	Free text

graphic questions, i.e., their experience in OSS and gender identities. Next, we asked participants to answer the following questions from the perspectives of both a PR reviewer and a PR author: (1) Have they encountered a no-response review comment? (2) If yes, how frequently do they encounter no-response comments? (3) What do they think could be the reasons for no-response comments? In this question, we intuitively give six options based on the category analysis of no-response comments and an 'other' option allowing them to add new thoughts. In the last, we invite participants to share their thoughts on the impact of no-response comments on OSS development and potential ways to address this issue. Table 3 shows the specific questions we asked and the optional answers.

3.3.2 Object Selection

Since we intend to ask participants to reply to our survey both from the developer and code reviewer perspectives, the objects we are going to invite should have rich OSS contributions, which can, to some extent,to some exten,t ensure he/she is both a developer and a code reviewer. Also, to improve the response rate, we target the developers who have had OSS development activities recently. Specifically, we select survey objects as follows:

- We cloned the top 100 Java projects on GitHub ranked by the number of stars and identified developers who had made commits within the past year.
- For each project, we ranked the developers based on the number of commits they made in the past year.
- We selected the top ten contributors for each project. Besides, considering non-core contributors may have a deeper understanding of ignored code reviews, we randomly selected ten contributors from the remaining developers per project. It should be noted that if a repository has fewer than 20 contributors, we select all the contributors from that repository.
- To prevent sending the same survey multiple times to the same developer, we applied a widely-used method [41] to merge developer identities.

Based on the aforementioned steps, we identified a total of 1,359 unique Java developers. We then searched their public emails, e.g., posted on their GitHub profiles, homepages, or curriculum vitae, and invited them to participate in our survey by sending emails. We finally sent 544 email invitations.

Within a month of waiting, we received 45 responses, resulting in a response rate of 8.3% $\left(\frac{45}{544}\right)$, which is comparable to existing studies in software engineering [42], [43]. More than half of all respondents (28 out of 45, 62.2%) have been involved in OSS for 1 to 10 years. Eleven individuals have been involved in OSS for over 10 years, while only six have been involved for less than a year. The majority of participants are male (43 out of 45). We analyzed all responses to the two open-ended questions by conducting a thematic analysis (detailed steps are similar to Sec. 3.2.2). Through this process, we found agreement in the final set of categories of responses listed in Sec. 4.3.

This article has been accepted for publication in IEEE Transactions on Software Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSE.2026.3681776

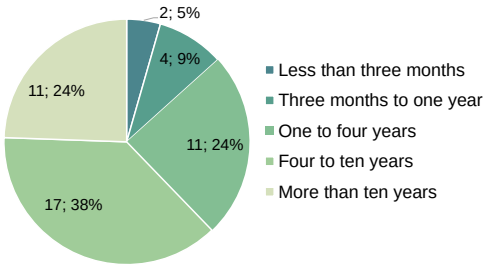

Fig. 3: Respondents' OSS experience.

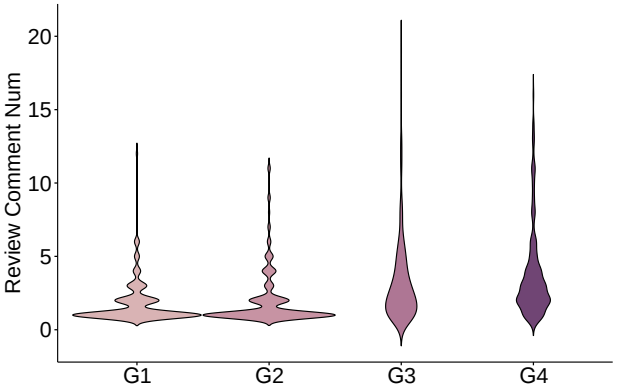

Fig. 4: Distribution of review comment numbers in the four project groups.

4 Results

This section presents the results, including the prevalence of no-response comments and possible negative impacts, (Sec. 4.1), a taxonomy of no-response comments (Sec. 4.2), and developers' perspectives towards the lack of response to review comments (Sec. 4.3).

4.1 RQ1: How Prevalent Are Review Comments Being Overlooked?

As introduced in Sec. 3.2.1, we randomly selected 10 PRs from each of the 80 projects and manually annotated them to find whether any no-response comments existed. The 80 projects are stratified sampled from the four groups of projects classified by their PR number quartiles (see details in Sec. 3.1). Some small repositories have fewer than 10 PRs that have at least one comment from reviewers. As a result, we selected 760 PRs that have 2,104 comments in total. Figure 4 shows the distributions of comment numbers in the 760 PRs selected from the four project groups. We can see that projects with more PRs tend to have more review comments. The median comment number of all the PRs is two. The reason for the small number might be that the stratified sampling method we adopted included some small repositories that have no formal code review process or do code review offline.

Using the method described in Sec. 3.2.1, we annotated the 2,104 comments and identified 712 (34%) comments that were not replied to. Figure 5 shows the distribution of no-response review comments among all the annotated ones. It is worth noting that not all comments from reviewers

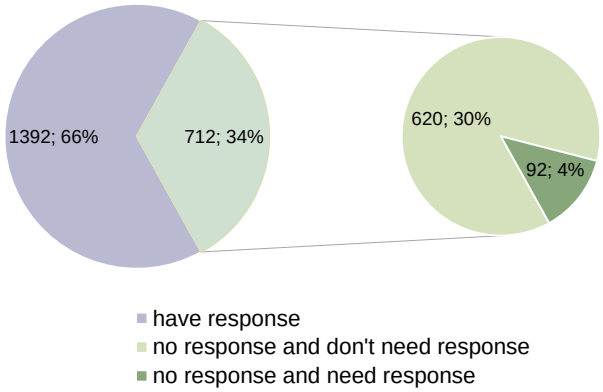

Fig. 5: Proportion of no-response comments of the 2,104 annotated comments.

need feedback, such as closing remarks like 'Good job!'. In this study, we *did not consider* this kind of comment as a no-response one. Through our manual analysis of the 712 comments, ***we identified 92 comments that needed a reply but were left unanswered, i.e., no-response comments.*** Specifically, the 92 no-response comments are from 57 PRs, which means that approximately 7.5% of PRs tend to have at least one review comment that received no response. The results demonstrate that there are indeed instances of no-response comments in the code review process but not in a common way.

We investigate whether PRs containing no-response comments tend to have a longer handling time, because such comments may reflect inadequate review interaction, potentially prolonging the review process. The median duration of the 753 analyzed PRs (excluding 7 open PRs) is 3 days, whereas the 57 PRs containing no-response comments have a markedly longer median duration of 20 days. A Mann–Whitney U test [44] showed that this difference is statistically significant (Cliff's $\delta = 0.347$, $p < 0.001$). However, we emphasize that this result should be interpreted as correlational rather than causal. Meanwhile, we found that 57.6% (53 out of 92) of review comments were posted more than a week after the PR creation date, and 38% (35 out of 92) were posted more than a month after the PR creation date. This suggests that one possible reason for unanswered review comments is that reviewers are unable to provide timely feedback on the PR. Delays in the review process may cause contributors to lose patience and interest in continuing to contribute.

We further investigate the acceptance outcomes of PRs containing no-response comments. We found that 68.4% (39 out of 57) of these PRs were not accepted, indicating that PRs with no-response comments have a substantially higher likelihood of being rejected. A chi-square test [45] confirmed that this difference in acceptance rates is statistically significant ($\chi^2 = 35.9$, $p < 0.001$), with PRs containing no-response comments being far less likely to be accepted (31.6%) compared to those without such comments (70.3%). While this result does not establish causality, it provides strong evidence that overlooked comments are correlated with negative project outcomes that may undermine review effectiveness and project sustainability.

Summary for RQ1: We identify 92 no-response comments in the 2,104 annotated ones, and approximately 7.5% of pull requests have review comments received no response from PR authors. It indicates that no-response comments exist but in an uncommon way when conducting code reviews. However, their presence is significantly associated with two negative review outcomes: pull requests with no-response comments experience significantly longer review lifecycles and are substantially less likely to be accepted than those without such comments. While overlooking one or two comments may not cause serious problems in each code review, the accumulation of no-response comments can lead to considerable waste and pose risks to the productivity of OSS development.

4.2 RQ2: What Types of Review Comments Have Been Ignored?

We apply thematic analysis [36] to analyze the identified no-response comments described in Sec. 3.2.2 and categorize eight types, which can be divided into two major categories, i.e., Review inquiry and PR management. The Review inquiry category includes four subcategories: Give suggestions about code implementation, Understand the motivation and code implementation, Point out implementation issues, and Additional task requests. The PR management category includes PR status checks, PR merge conflict notifications, and Reject PR with uncertain reasons. We introduce the two categories of no-response comments, including the subcategories, as follows. The purpose of this categorization is to characterize the comments that did not receive a response, rather than to assert that comments within these categories face a higher probability of being ignored.

4.2.1 Review Inquiry

Review inquiry indicates queries about code changes proposed during the code review process. These queries typically involve providing suggestions for code modifications, requesting the addition of tests, identifying existing issues, or offering improvements to the codebase. Code reviews are geared towards ensuring the quality and integrity of the codebase by addressing specific technical issues and potential improvements [5]. When review inquiries go unanswered, opportunities to improve code quality and address potential issues may be missed. We identified the following four types of Review inquiry.

RI1 **Give suggestions about code implementation** (#36): Comments in this category offer suggestions for improving the implementation. This includes suggestions for more efficient algorithms, improved data structures, or alternative coding approaches that would improve the readability, maintainability, or performance of the code. For example, *"Instead of doing this, it may be better to have a subclass for the case where there are two values. Also, reset would have to reset secondValue too."*

RI2 **Ask questions regarding code implementation** (#18): Comments in this category seek further clarification on both the motivation behind the code changes and the implementation details. This includes questions about the reasoning behind certain design decisions, the specifics of the algorithm, or particular aspects of the code implementation. Reviewers may inquire *"What's the point of adding three classes and never calling the register method?"*

RI3 **Point out implementation issues** (#7): Comments in this category highlight specific issues or problems related to the implementation of code changes. This includes pointing out bugs, syntax errors, or design issues that compromise the overall quality or functionality of the code changes. For example, *"Hi, I don't think you want to have the @Before annotations here as they indicate that the function should be run before each test. The result is that each "test" function is being run multiple times."*

RI4 **Additional task requests** (#6): Comments in this category request additional tasks to ensure the functionality and reliability of the code changes. For example, *"It is better to test when resultSet.getString('COLLATION') is null."*

4.2.2 PR Management

This category refers to the comments that are focused on the administrative aspects of the code review process, ensuring that the PR is being appropriately progressed through the development pipeline. These inquiries include PR status checks, notifications of merge conflicts, and Reject PR with uncertain reasons. It is important to note that ignoring this kind of review comments can greatly lead to the 'close' of a pull request, i.e., not accepting the code change. We identified 25 comments belonging to this category. We introduce the three subcategories of PR management as follows:

PM1 **PR status checks** (#6): Comments in this category typically pertain to the PR status and whether relevant issues have been resolved before being merged into the main codebase. This includes formatting inconsistencies and checking if the PR author is still updating the PR, which ensures the code is up to par with project standards. For example, *"@PR_author Hi what's going on with this PR? Do U want to continue or close it?"*

PM2 **PR merge conflict notifications** (#6): These comments alert the PR author of conflicts when attempting to merge the PR with the main branch. Typically, these conflicts are caused by other collaborators modifying the same code modules, such as environment variables, in the main branch, requiring additional attention and resolution. For example, *"good idea. There are merge conflicts. We should probably put a hard link and/or instructions for setting environment variables. I reckon a search is okay though."*

PM3 **Reject PR with uncertain reasons** (#13): These comments indicate a decision to reject the PR, but they are accompanied by uncertain reasons for the rejection. Because of uncertainties, responding to them may still lead to the acceptance of the PR. For example, *"You should better pass lint check before commit."* The review comment regarding the issues did not receive a response, and then the PR reviewer closed the PR.

This article has been accepted for publication in IEEE Transactions on Software Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSE.2026.3681776

Summary for RQ2: There are two main types of no-response comments: Review inquiry and PR management. The most common no-response comments are about asking questions or giving suggestions regarding code implementation. Neglecting these comments could either lead to the failure of the Pull requests or missing opportunities for optimizing code quality and addressing potential issues. The results can help developers and researchers have a better understanding of PR comments that were overlooked.

4.3 RQ3: How Do Developers Perceive No-response Comments?

As introduced in Sec. 3.3, we surveyed developers about their practical perspectives on no-response comments and received 45 responses. The perspectives of developers are mainly divided into four categories: the prevalence of no-response comments from their experience, the reasons for no-response comments, the impact of no-response comments, and their suggestions to reduce this phenomenon. We use the notation 'P{num}' to refer to a specific participant's responses, with the number inside the braces indicating the participant ID.

4.3.1 Prevalence of no-response comments

Table 4 illustrates whether developers have encountered "no-response" comments and the frequency within the past year. Note that we ask developers to answer the same questions based on their experience of the two roles, i.e., being reviewers and developers. Table 3 displays the specific content of the questionnaire. From Table 4, we can see that as PR reviewers, 64.4% (29 out of 45) of participants reported experiencing no-response comments. Among the 29 'Yes' reviewers, two reviewers reported more than 12 times, 48.3% reported a frequency between 4 and 6 times, and 44.8% of them reported a frequency of fewer than 4 times within a year. Since we asked about the frequency of comments being neglected they have experienced over the past year, the majority frequencies, i.e., more than four times, indicate that review comments encountering no response is not a rare phenomenon in practice. On the other hand, 24.4% of respondents indicated they had never experienced no-response comments. Additionally, five participants could not recall whether their comments were being ignored.

Similarly, as PR authors, 28.9% (13 out of 45) of respondents admitted to having not replied to some review comments on their pull requests. For those developers who have overlooked review comments, 84.6% admit that they have ignored review comments less than 4 times in the past year, while 15.4% stated that they have ignored review comments between 4 and 6 times. A significant majority of 66.7% reported they have never ignored review comments. Approximately 4.4% of participants could not remember if they had missed review comments.

4.3.2 Reasons for no-response comments

To understand the underlying reasons behind no-response comments, we asked for developers' perspectives from both PR author and reviewer roles. Figure 6 shows the categorized reasons and their respective proportions.

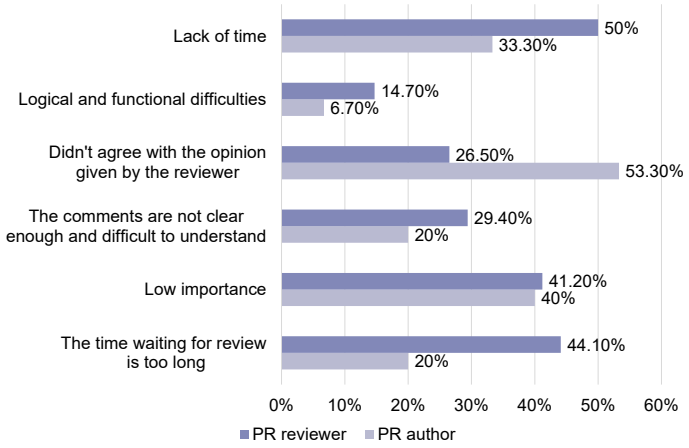

Fig. 6: Reasons given by developers towards the existence of no-response review comments.

- **Lack of time**: Highlighted by 33.3% of PR authors and 50% of PR reviewers, the lack of time is a significant factor contributing to no-response comments. This reason reflects the time constraints faced by contributors, emphasizing the need for possibly more efficient review processes within projects.
- **Didn't agree with the opinion given by the reviewer**: This reason is predominantly cited by PR authors, constituting 53.3% of the group, compared to 26.5% of PR reviewers. It reveals a significant disparity in perspectives, suggesting that disagreements with reviewers' opinions are a major factor for authors not responding to comments. This indicates a potential need for conflict resolution mechanisms within the review process.
- **Low importance**: Both PR authors and reviewers agree on this point, with 40% of authors and 41.2% of reviewers citing low importance as a reason for no-response comments. PR authors might be more inclined to ignore comments they deem low-importance.
- **Time waiting for review is too long**: This reason accounts for 20% of PR authors and a significantly higher 44.1% of PR reviewers. It suggests that a notable fraction of reviewers believe that the reason hindering the PR author from further engaging in the PR is the delay in the review process, leading to no-response comments. Conversely, a smaller portion of authors perceive this as a major issue, possibly indicating a cognitive bias between the two groups of developers in terms of review time.
- **Comments are not clear enough and difficult to understand**: While 29.4% of PR reviewers felt that the comments were not clear enough and difficult to understand, a slightly lower percentage of 20.0% of PR authors shared this sentiment. This discrepancy suggests a perception difference between the two groups regarding the clarity and comprehensibility of comments. This recognition underscores the importance of clear communication in the review process and the challenges posed by ambiguous review comments.
- **Logical and functional difficulties**: Cited by a relatively small fraction, 6.7% of PR authors and 14.7% of PR

Table 4: Participants' responses to encountering no-response comments.

Role	Encountered	Num & Ratio	Frequency	Num & Ratio
Reviewer	Yes	**29 (64.4%)**	Less than 4 times	13 (44.9%)
	No	11 (24.4%)	4 to 6 times	**14 (48.3%)**
	Not sure	5 (11.1%)	More than 12 times	2 (6.9%)
Author	Yes	13 (28.9%)	Less than 4 times	**11 (84.6%)**
	No	**30 (66.7%)**	4 to 6 times	2 (15.4%)
	Not sure	2 (4.4%)	- -	- -

reviewers attribute no-response comments to logical and functional difficulties encountered. This reason points to the technical challenges that might hinder addressing certain comments, though it appears to be a less prevalent issue compared to others.

- **Other**: Beyond the structured response options, eight participants provided free-text explanations that reveal additional nuances behind no-response comments. From the reviewer perspective, P11 noted a social-psychological factor related to contributor motivation, stating that "Mostly they don't care beyond their name showing up in our repo as contributor." This highlights that, for some contributors, engagement in the review process may not be perceived as necessary once the contribution itself is visible. P2 pointed to limitations of the current platform, remarking that "GitHub isn't good at presenting multiple review comments (at once)," suggesting that tooling constraints can impede the visibility and handling of feedback, leading to unintentional neglect. On the author side, respondents emphasized that GitHub is not always the sole communication channel for addressing review feedback. P10 explained that "some reviews might be responded to through email, chat group, or other way, not GitHub," indicating that resolutions may occur externally and thus appear as no response within the PR interface. P33 pointed to developer inactivity as a cause, noting succinctly that "the author is no longer active," which implies that contributors have disengaged from the project or abandoned earlier PRs.

These free-text responses provide additional contextual factors, ranging from contributor motivation to platform limitations and external communication practices, which complement the structured reasons presented in Figure 6. These findings illuminate the differences in perceptions between PR authors and reviewers regarding giving no response to comments. Understanding these reasons is crucial for improving communication and collaboration within code review, ultimately enhancing software development.

4.3.3 Impact of no-response comments

To understand the impact of no-response comments on OSS development, this survey prepared a free-text question to address this question. Out of the 45 responses received, 35 participants shared their thoughts on the potential impact of ignored review comments. Through manual analysis, we have identified five major categories.

- **Prevent the PR from being merged**: Eight participants explicitly stated that if review comments are not addressed, they would reject the PR from being merged. This leads to a waste of resources, consuming time and effort for both the PR reviewer and the PR author. For example, P38 mentioned, *"If no responses received, I would block the PR from being merged."*
- **Detrimental to collaboration**: Eight participants stated that this phenomenon can detrimentally affect collaboration within the community. Reviewers, feeling their contributions and efforts are undervalued, may become disinclined to participate in future review processes. This reluctance undermines the collaborative spirit essential for the success and growth of OSS projects. For example, P4 stated, *"Eventually, cabals form where the Authors only consider certain reviewers as worth considering. This eventually discourages more reviewers from participating in the review process."* Similarly, P15 mentioned, *"Enthusiasm for this project will decrease."*
- **Slow down the development progress and increase the burden on developers**: Seven participants stated that ignored comments can slow down the development progress. Unresolved discussions or suggestions could lead to additional workload for developers, as they might need to revisit these issues later in the development cycle. This not only delays the current project timeline but also increases the burden on developers, potentially affecting the overall productivity of the team. For example, P18 mentioned, *"I think this will slow down the development progress and cause the PR to not be merged well. Some developers will manually fix it and then merge the PR, but this will increase the burden on developers."* Another potential impact is that it may discourage other contributors from working on the same topic. As P21 stated, *"No other contributor will start working on this topic as long as there is an open PR of someone else."*
- **Some critical issues will remain unresolved**: A critical concern highlighted by the survey is that some key issues may remain unresolved because of ignored comments. Essential feedback or issues that require attention could be overlooked, leading to suboptimal code quality or even security vulnerabilities within the project. Five participants mentioned this potential consequence. For example, P14 stated, *"As a result, some problems cannot be solved in time."*
- **Diminish reputation within the OSS community**: The practice of not replying to comments can tarnish a contributor's reputation within the project. Active and responsive engagement is often seen as a hallmark of a committed and reliable developer. Insufficient of feedback can be perceived as a lack of interest or respect for the community's collaborative efforts, potentially affecting one's standing and opportunities for future

contributions. Two participants mentioned this impact. For example, P32 stated, *"Not replying to comments may be because you are busy, but not replying to comments will reduce your trust in the project."*

Ignored review comments in OSS development can have various negative impacts, including harming development efficiency and code quality. Addressing review comments promptly is crucial for maintaining a thriving OSS community.

4.3.4 Potential actions in the future

The survey yielded a diverse range of suggestions for reducing the incidence of no-response comments in PRs. Out of the 45 responses received, 33 participants provided suggestions to address this issue. By analyzing these responses, five key themes emerged, which can be broadly categorized into improvements in communication and collaboration, the establishment of guidelines, the use of automation tools, the importance of community culture, and enhancements in interface and usability.

- **Communication and collaboration enhancements**: Seven participants emphasized the need for clearer communication and better collaboration practices in the review process. Reviewers should ensure that their comments are clear to facilitate smoother communication with PR authors. As P21 stated, *"I think it is important that the comments are clear and have good arguments if and why changes are needed."* Additionally, if reviewers find that their comments have not been addressed, they should proactively reach out to the PR author. P41 suggested, *"Try to reach out to the reviewers through community channel or DM them asking if they are available."*
- **Clarify the rules involved in the review process**: Thirteen participants provided suggestions related to rule-setting, including clearer contributor guidelines, more standardized development documentation, and effective collaboration guidelines. For example, P11 suggested, *"Explicit contributor guideline."*and P23 recommended, *"set more effective collaboration guidelines."* Among them, eight participants explicitly stated that if a PR author ignores code review feedback or fails to follow established guidelines, they would refuse to merge the PR. For instance, P38 proposed, *"We should set rules and write them into the development documentation. If the rules are not met, we will directly close the PR."*
- **Foster supportive community culture**: Four participants highlighted the significance of fostering a supportive community culture where review remarks are taken seriously and not ignored. As P40 stated,*"Foremost, there needs to be a common understanding in the team that review remarks should be taken seriously and not discarded silently."* Additionally, a positive community culture can contribute to a better social atmosphere. P43 mentioned, *"What the community can do is to be as friendly and patient as possible to help the project succeed together!"*
- **Automation**: The use of automation to improve the review process was frequently mentioned. Four participants mentioned the use of automation tools to check whether review comments have been addressed. For example, P31 suggested, *"It would help if a bot could check whether some changes are made to fix the comments."* Similarly, P10 suggested, *"What some teams do is to set each suggestion as a Github task, and to then have automation that ensures that each task is completed or at least closed manually before the pull request can be merged."*
- **Interface and Usability Improvements**: A few suggestions focused on improving the user interface for presenting open review comments. Three participants made similar suggestions, such as introducing a dedicated code review interface. P2 stated, *"Better UI for presenting (open) review comments - e.g. in a dedicated section rather than inline with the rest of the discussion."* This could help in making unresolved comments more visible and thereby reducing the likelihood of them being overlooked. Additionally, one suggestion was to allow developers to set their usual online hours on their personal profiles to facilitate a smoother review process. P18 mentioned, *"To ensure that each PR can be merged quickly without problems, the usual online time can be set on the GitHub profile. I suggest that the maintainer urge the PR author to complete it during his online time."*

Summary for RQ3: Both reviewers (64%) and developers (29%) have experienced no-response reviews. Possible reasons can be prolonged review times, perceived comment insignificance, lack of clarity, disagreement with opinions, and time constraints. Ignored reviews can impede collaboration, slow down development, cause project bottlenecks, and erode trust within the team and developer community. To mitigate these impacts, it is crucial to have active engagement with reviewer comments, clear collaboration guidelines, and a culture of open communication.

5 Discussion

Code reviews are of pivotal importance to facilitate code quality and knowledge sharing. In this section, we discuss the implications of no-response comments for developers and researchers based on our findings.

5.1 No-response comments deserve more attention

Our study found that no-response comments exist but in an uncommon way, when we consider all the Java OSS projects in OSS by using stratified sampling. This phenomenon may be more prevalent in active projects with more newcomers. To shed some light on this, we performed an additional manual check on the project with the largest number of PRs in our dataset, i.e., elasticsearch, a distributed search and analytics engine built for speed, scale, and AI applications[3]. We randomly sampled 100 PRs from that project and manually examined all 382 review comments. The first two authors double-checked the labeled results and discussed the resolved inconsistency. We found that 15 of the 100 PRs (15%, higher than 7.5%) contained at least one no-response comment. The newly added annotated data and classification results can be found in our online appendix [38]. This elevated rate in a highly active repository suggests that

3. https://www.elastic.co/elasticsearch

projects with more newcomers and high maintainer load may be more susceptible to no-response comments. In addition, the observed patterns are consistent with our primary dataset, suggesting that the phenomenon is not limited to the initial project set. The Elasticsearch check, therefore, complements our cross-project findings and echoes the patterns reported in the survey. Qualitative research indicates that reviewers encounter such comments significantly more frequently in their daily work than 7.5%: 64.4% of reviewers reported having encountered this situation in the past year, with over 90% of those respondents encountering it between one and six times. Despite the modest overall fraction of no-response comments, their frequency in active projects and their recurrence in developers' experience indicate that these comments are not negligible and deserve targeted tooling and process attention. Based on our findings, future work could systematically investigate the characteristics of no-response comments in a more controlled manner, comparing with reviews that have responses, for instance.

5.2 Negative effects of no-response comments

Our findings reveal that even though no-response comments constitute a relatively small fraction of all review comments, their effects on software projects are both measurable and consequential. As shown in Sec. 4.1, PRs containing at least one no-response comment exhibited substantially longer review lifecycles and were significantly less likely to be accepted, compared to PRs without such comments. These results suggest that overlooked reviewer input may impede the timely resolution of issues, disrupt review momentum, and ultimately reduce the likelihood that valuable contributions are integrated into the project. The qualitative feedback from our survey further reinforces these observations. Some developers stated directly that a PR should not be merged if the author does not respond to review feedback, as illustrated by P17: "... I think if the PR authors do not respond to reviews, this PR should not be approved." Others emphasized that the presence of no-response comments can slow development progress, erode trust and cooperation within the community, and discourage reviewers from investing effort in future reviews. As P23 explained: "When pull request authors don't respond to reviews, it can slow down progress, lead to code quality issues, and create frustration among reviewers. It may also disrupt communication and affect the overall health of the project." Therefore, despite their modest frequency, the cumulative and long-term impact of no-response comments on the sustainability and health of OSS projects is substantial and warrants further research attention. An important direction for future research is to examine whether no-response comments can cause downstream software quality problems beyond PR-level outcomes. Comments left unanswered may indicate that potential issues raised during review were not fully resolved, which could in turn affect the correctness, maintainability, or robustness of software. Investigating this possibility would help clarify whether no-response comments are not only a review-process concern, but also a signal of technical risk.

5.3 Communication between PR reviewer and PR author is important

Communication between developers and reviewers is crucial in the software development process. If there are disagreements between the two parties, the project may enter an unhealthy state, leading to serious issues. Coupled with our exploration of reasons in RQ3, "Didn't Agree with the Opinion Given by the Reviewer" (26.5% in PR reviewer and 53.3% in PR author), further confirms this phenomenon. Over half of developers believe that disagreement with the reviewer's opinion is a primary reason for no-response comments. On the other hand, according to our survey results, PR reviewers and PR authors have different perceptions of the frequency of no-response comments. This indicates that many no-response comments are overlooked by PR authors without being aware of them. Therefore, we believe that strengthening communication and addressing these disagreements can significantly reduce the occurrence of no-response comments. For example, code review processes (such as GitHub PRs) can introduce more convenient and efficient communication mechanisms or better conflict resolution mechanisms. An interesting open question concerns whether the observed discrepancies between author and reviewer perspectives are primarily role-driven or individual-driven. In practice, many developers act as both authors and reviewers across projects. To keep our survey anonymous and ease ethical concerns, we did not collect identifiable information that would allow us to link responses across roles. Future work could examine whether differences stem from situational role expectations or stable individual tendencies.

5.4 Incentive misalignment and socio-psychological factors

While Sec. 4.3.2 summarizes the reasons reported by participants, such as lack of time, unclear comments, or disagreement, our results and the reviewer example highlight a deeper mechanism that is not fully visible in the responses: incentive misalignment between volunteer contributors and project maintainers. Contributors often submit pull requests to address their own needs, and once their immediate goal has been met, their motivation to continue investing time diminishes rapidly. Several empirical observations from our study are consistent with this interpretation. 38% of no-response comments were posted more than one month after PR creation, long after the contributor's attention or interest may have shifted. Survey responses referenced shifting priorities or insufficient perceived value in continuing the discussion. P11 noted a social-psychological factor related to contributor motivation, stating that "Mostly they don't care beyond their name showing up in our repo as contributor." This indicates that for some contributors, once their contributions themselves are visible, participating in review conversations may seem unnecessary. Importantly, the most frequently selected reason, "lack of time", may represent a polite proxy for deeper issues such as diminishing personal benefit, perceived unfairness of late requests, or the absence of reciprocal value. From a social-exchange perspective, contributors may feel little obligation to respond when the return of additional work is unclear, especially

when feedback arrives long after their contribution was made.

Therefore, interventions must address the broader motivational context of OSS collaboration. Future work could incorporate interviews or longitudinal studies to more directly capture these socio-psychological processes. Experimental evaluation of incentive mechanisms (e.g., reputation systems, lightweight rewards) or AI-mediated compromise suggestions may help identify strategies that realign incentives and reduce breakdowns in reviewer–author coordination. One promising direction is to explore AI-based mediation techniques to help realign incentives and reduce friction. For example, intelligent assistants could promptly notify both parties, such as via email, after a contributor creates a PR to expedite the code review process. Alternatively, intelligent assistants could propose compromises between reviewers and authors, automatically generate corrective patches for minor issues. Such a system could act as an intermediary, maintaining mutual benefit without placing additional cognitive or emotional burden on either party.

6 Threats to Validity

6.1 Internal Validity.

The first internal threat lies in the criteria for selecting repositories, which required at least 10 PRs and 3 contributors. While these thresholds were set to avoid inactive projects and the issue of reviewers evaluating their own PRs, they could exclude smaller or less active repositories that might still offer valuable insights. Additionally, this selection may limit the generalizability of the findings to the broader range of Java open-source projects. Future work should consider expanding the selection criteria to include a wider variety of repositories.

During the data annotation phase, manual labeling of review comments poses a threat to subjective validity. To minimize the errors stemming from this threat, two authors conducted manual annotations independently and enlisted an experienced colleague to assist in judgment. The high level of agreement (0.94) indicates that the manual annotations by the two authors are quite reliable. For differently labeled review comments, the two authors engaged in multiple discussions with the experienced colleague and subsequently reached a consensus.

6.2 External Validity.

6.2.1 Sampling Strategy and Dataset Representativeness

Studying only Java projects limits external validity. Projects that use only one programming language have limitations and may not fully reflect patterns of unresponsive comments across the entire open-source community. In projects using other programming languages, code review patterns may differ. This paper does not explore potential differences in unresponsive comment patterns. Our research findings can serve as a foundation, and future studies should investigate a wider range of projects, including sampling different languages or conducting more comprehensive project sampling to gain deeper insights into comments overlooked in modern code reviews.

Additionally, during the data collection phase, our study analyzed 80 repositories, selecting 10 pull requests (PRs) from each. Although limiting the number of PRs per repository may introduce bias—since a small subset might not fully reflect the review practices of a specific project, we aimed to ensure diversity and representativeness at the dataset level. To achieve this, we adopted a stratified sampling strategy that covered repositories of varying sizes, domains, and activity levels, resulting in a total of 760 PRs that collectively captured a broad spectrum of review contexts. Thus, while individual repositories may exhibit localized biases, the overall dataset maintains strong heterogeneity and cross-project coverage. This design choice reflects a practical balance between analytical depth and scope, given the substantial manual effort required for fine-grained annotation. To evaluate whether this per-repository depth materially affected our findings, we conducted an additional targeted check on the repository with the most PRs in our dataset (i.e., elasticsearch). We randomly sampled 100 PRs from that project and manually analyzed all review comments. The manual review did not reveal new categories of no-response comments beyond those already identified in our original coding schema. This suggests that our category of no-response comment captures the variability of no-response behavior, and that increasing the per-project depth would not materially change our findings.

6.2.2 Review Comment Source

Studying overlooked code review patterns solely through exploring review comments in pull requests has its limitations. In the open-source software development process, developers can communicate through various means beyond pull requests, such as live chat, email, and issue discussions. These alternative methods have not been effectively explored in this study, and different modes of communication may harbor distinct patterns of no-response comments. In future work, we will explore the possible existence of no-response comments in these other communication channels.

6.2.3 Survey Sampling Bias

A potential source of bias arises from the relatively small number of survey respondents, i.e., 45 developers in our study. Although this limits the statistical generalizability of our findings, we mitigated sampling bias by recruiting participants from the top 100 Java projects on GitHub, selecting both core contributors (top 10 by commit count) and non-core contributors (10 randomly chosen). This design aimed to capture a diverse range of experience levels and project roles. The 8.3% response rate is comparable to response rates reported in prior software engineering survey studies [42], [43]. Nonetheless, the modest response rate means our sample may not fully represent the broader OSS developer community. To reduce overinterpretation, we treat the survey as complementary qualitative evidence that contextualizes our quantitative analyses. Future research could enhance external validity through larger-scale surveys and multiple recruitment channels, such as mailing lists and community forums.

6.2.4 Scope Limitation: OSS versus Non-OSS Contexts

Our study focuses exclusively on OSS projects, where participation is voluntary, and contributors have considerable autonomy in deciding whether to respond to review comments. This contextual feature likely moderates the impact of no-response comments: while we found that only 7.5% of OSS pull requests contain such comments, this relatively low prevalence may reflect the informal and flexible nature of open collaboration. In contrast, in non-OSS or industrial environments, where review processes are often mandatory and directly tied to product release cycles or performance evaluation, unaddressed comments could represent a far more critical issue. Ignoring review feedback in these settings might lead to quality assurance risks, schedule delays, or communication breakdowns within teams. However, due to the proprietary and confidential nature of closed-source repositories, we were unable to replicate our analysis in industrial contexts. As a result, the generalizability of our findings beyond OSS projects remains limited. We explicitly acknowledge this boundary as a threat to external validity and suggest that future research extend the investigation to organizational and enterprise software environments to assess whether the magnitude and implications of no-response comments differ under formal development governance.

7 Conclusion

In this paper, we explored the prevalence and patterns of comments that require a response but remain unanswered during the code review process, as well as the developers' perspectives towards this phenomenon. Our findings reveal that 7.5% of pull requests contain no-response review comments. Beyond prevalence, our statistical analyses show that the presence of no-response comments is significantly associated with negative review outcomes, including substantially longer review lifecycles and lower PR acceptance rates. Moreover, an additional re-annotation study on a large, highly active project reveals an even higher proportion of PRs containing no-response comments, further underscoring the practical relevance and necessity of systematically studying this phenomenon. A thematic analysis uncovered seven categories of no-response comments, with the majority falling into two main groups: Review inquiry and PR management. These overlooked comments represent missed opportunities for improving code quality and project outcomes. Consistent with prior observation, our survey results indicate that no-response comments are not rare in practice, as 64.4% of participants reported encountering no-response comments during their day-to-day development activities. The reasons for the lack of response to these comments include prolonged review times, perceived insignificance of comments, lack of clarity, disagreement with opinions, and time constraints. This neglect can hinder collaboration, slow development progress, and weaken trust within the team and broader developer community. By fostering active engagement, timely responses, and open communication, OSS communities can mitigate the risks revealed in this study. More broadly, we hope this work encourages the community and researchers to treat responsiveness as a first-class property of code review processes, and to design tools, norms, and incentive mechanisms that better sustain long-term, large-scale open collaboration.

Data Availability

To facilitate replication and possibly assist other future work, we provide the data and retrieved materials used in this study in our online appendix [38].

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grant No. 62572048, 62232003, 62332001, and 62272445).

References

[1] A. F. Ackerman, P. J. Fowler, and R. G. Ebenau, "Software inspections and the industrial production of software," in *Proc. of a Symposium on Software Validation: Inspection-Testing-Verification-Alternatives*. USA: Elsevier North-Holland, Inc., 1984, p. 13–40.

[2] A. Ackerman, L. Buchwald, and F. Lewski, "Software inspections: an effective verification process," *IEEE Software*, vol. 6, no. 3, pp. 31–36, 1989.

[3] A. Bosu, J. C. Carver, C. Bird, J. Orbeck, and C. 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*, vol. 43, no. 1, pp. 56–75, 2017.

[4] M. Fagan, *Design and Code Inspections to Reduce Errors in Program Development*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 575–607. [Online]. Available: https://doi.org/10.1007/978-3-642-59412-0_35

[5] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 712–721.

[6] J. Cohen, "Modern code review," *Making Software: What Really Works, and Why We Believe It*, pp. 329–336, 2010.

[7] P. C. Rigby and C. Bird, "Convergent contemporary software peer review practices," in *Proceedings of the 2013 9th joint meeting on foundations of software engineering*, 2013, pp. 202–212.

[8] I. Steinmacher, T. Conte, M. A. Gerosa, and D. Redmiles, "Social barriers faced by newcomers placing their first contribution in open source software projects," in *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, ser. CSCW '15. New York, NY, USA: ACM, 2015, pp. 1379–1392.

[9] M. Zhou, Q. Chen, A. Mockus, and F. Wu, "On the scalability of linux kernel maintainers' work," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 27–37.

[10] X. Tan, M. Zhou, and B. Fitzgerald, "Scaling open source communities: An empirical study of the linux kernel," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1222–1234. [Online]. Available: https://doi.org/10.1145/3377811.3380920

[11] A. Bosu, M. Greiler, and C. Bird, "Characteristics of useful code reviews: An empirical study at microsoft," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 146–156.

[12] M. Hasan, A. Iqbal, M. R. U. Islam, A. I. Rahman, and A. Bosu, "Using a balanced scorecard to identify opportunities to improve code review effectiveness: An industrial experience report," *Empirical Software Engineering*, vol. 26, pp. 1–34, 2021.

[13] B. S. Meyers, N. Munaiah, E. Prud'hommeaux, A. Meneely, J. Wolff, C. O. Alm, and P. Murukannaiah, "A dataset for identifying actionable feedback in collaborative software development," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2018, pp. 126–131.

[14] M. M. Rahman, C. K. Roy, and R. 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 (MSR)*. IEEE, 2017, pp. 215–226.

[15] R. Tufano, L. Pascarella, M. Tufano, D. Poshyvanyk, and G. Bavota, "Towards automating code review activities," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021, pp. 163–174.

[16] R. Tufano, S. Masiero, A. Mastropaolo, L. Pascarella, D. Poshyvanyk, and G. Bavota, "Using pre-trained models to boost code review automation," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 2291–2302.

[17] Y. Zhang, Y. Zhang, Z. Sun, Y. Jiang, and H. Liu, "Laura: Enhancing code review generation with context-enriched retrieval-augmented llm," in *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2025, pp. 2983–2995.

[18] M. Beller, A. Bacchelli, A. Zaidman, and E. Juergens, "Modern code reviews in open-source projects: Which problems do they fix?" in *Proceedings of the 11th working conference on mining software repositories*, 2014, pp. 202–211.

[19] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "An empirical study of the impact of modern code review practices on software quality," *Empirical Software Engineering*, vol. 21, pp. 2146–2189, 2016.

[20] D. Spadini, G. Çalikli, and A. Bacchelli, "Primers or reminders? the effects of existing review comments on code review," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 1171–1182.

[21] O. Kononenko, O. Baysal, L. Guerrouj, Y. Cao, and M. W. Godfrey, "Investigating code review quality: Do people and participation matter?" in *2015 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 2015, pp. 111–120.

[22] O. Kononenko, O. Baysal, and M. W. Godfrey, "Code review quality: How developers see it," in *Proceedings of the 38th international conference on software engineering*, 2016, pp. 1028–1038.

[23] P. Thongtanunam, R. G. Kula, A. E. C. Cruz, N. Yoshida, and H. Iida, "Improving code review effectiveness through reviewer recommendations," in *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering*, 2014, pp. 119–122.

[24] M. B. Zanjani, H. Kagdi, and C. Bird, "Automatically recommending peer reviewers in modern code review," *IEEE Transactions on Software Engineering*, vol. 42, no. 6, pp. 530–543, 2015.

[25] G. Rong, Y. Zhang, L. Yang, F. Zhang, H. Kuang, and H. Zhang, "Modeling review history for reviewer recommendation: a hypergraph approach," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1381–1392. [Online]. Available: https://doi.org/10.1145/3510003.3510213

[26] J. Tsay, L. Dabbish, and J. Herbsleb, "Influence of social and technical factors for evaluating contribution in github," in *Proceedings of the 36th international conference on Software engineering*, 2014, pp. 356–366.

[27] T. Dey and A. Mockus, "Effect of technical and social factors on pull request quality for the npm ecosystem," in *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2020, pp. 1–11.

[28] X. Zhang, Y. Yu, G. Gousios, and A. Rastogi, "Pull request decisions explained: An empirical overview," *IEEE Transactions on Software Engineering*, vol. 49, no. 2, pp. 849–871, 2022.

[29] A. Krutauz, T. Dey, P. C. Rigby, and A. Mockus, "Do code review measures explain the incidence of post-release defects? case study replications and bayesian networks," *Empirical Softw. Engg.*, vol. 25, no. 5, p. 3323–3356, Sep. 2020. [Online]. Available: https://doi.org/10.1007/s10664-020-09837-4

[30] E. D. Berger, C. Hollenbeck, P. Maj, O. Vitek, and J. Vitek, "On the impact of programming languages on code quality: A reproduction study," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 41, no. 4, pp. 1–24, 2019.

[31] S. O. Community, "2023 developer survey," https://survey.stackoverflow.co/2023/#most-popular-technologies-language, 2023.

[32] Y. Tian, Y. Zhang, K.-J. Stol, L. Jiang, and H. Liu, "What makes a good commit message?" in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 2389–2401. [Online]. Available: https://doi.org/10.1145/3510003.3510205

[33] J. Neyman, "On the two different aspects of the representative method: the method of stratified sampling and the method of purposive selection," in *Breakthroughs in statistics: Methodology and distribution*. Springer, 1992, pp. 123–150.

[34] O. Dabic, E. Aghajani, and G. Bavota, "Sampling projects in github for MSR studies," in *18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021*. IEEE, 2021, pp. 560–564.

[35] GitHub, "Github rest api," https://docs.github.com/en/rest, 2024.

[36] D. S. Cruzes and T. Dyba, "Recommended steps for thematic synthesis in software engineering," in *2011 International Symposium on Empirical Software Engineering and Measurement*. IEEE Computer Society, 2011, pp. 275–284. [Online]. Available: https://doi.org/10.1109/ESEM.2011.36

[37] G. Gousios, M. Pinzger, and A. v. Deursen, "An exploratory study of the pull-based software development model," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 345–355. [Online]. Available: https://doi.org/10.1145/2568225.2568260

[38] Y. Li, Y. Zhang, Q. Zeng, L. Shi, X. Tan, T. Wang, Y. Jiang, and H. Liu, "Online appendix of 'an empirical study of overlooked code reviews in open source software projects'," https://figshare.com/s/15772c55fbbc6a31c1a2, 2025.

[39] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and psychological measurement*, vol. 20, no. 1, pp. 37–46, 1960.

[40] J. S. Molléri, K. Petersen, and E. Mendes, "Survey guidelines in software engineering: An annotated review," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2961111.2962619

[41] J. Zhu and J. Wei, "An empirical study of multiple names and email addresses in oss version control repositories," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 409–420.

[42] M. Fan, Y. Zhang, K.-J. Stol, and H. Liu, "Core developer turnover in the rust package ecosystem: Prevalence, impact, and awareness," *Proc. ACM Softw. Eng.*, vol. 2, no. FSE, Jun. 2025. [Online]. Available: https://doi.org/10.1145/3729392

[43] E. Smith, R. Loftin, E. Murphy-Hill, C. Bird, and T. Zimmermann, "Improving developer participation rates in surveys," in *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2013, pp. 89–92.

[44] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[45] K. Pearson, *On the Criterion that a Given System of Deviations from the Probable in the Case of a Correlated System of Variables is Such that it Can be Reasonably Supposed to have Arisen from Random Sampling*. New York, NY: Springer New York, 1992, pp. 11–28. [Online]. Available: https://doi.org/10.1007/978-1-4612-4380-9_2