Empirical Software Engineering manuscript No.
(will be inserted by the editor)

What Makes a Code Review Useful to OpenDev Developers? An Empirical Investigation

Asif Kamal Turzo · Amiangshu Bosu

Submitted: February 20, 2023 / Accepted: TBD

Abstract *Context:* Due to the association of significant efforts, even a minor improvement in the effectiveness of Code Reviews(CR) can incur significant savings for a software development organization.

Aim: This study aims *to develop a finer grain understanding of what makes a code review comment useful to OSS developers, to what extent a code review comment is considered useful to them, and how various contextual and participant-related factors influence its degree of usefulness.*

Method: On this goal, we have conducted a three-stage mixed-method study. We randomly selected 2,500 CR comments from the OpenDev Nova project and manually categorized the comments. We designed a survey of OpenDev developers to better understand their perspectives on useful CRs. Combining our survey-obtained scores with our manually labeled dataset, we trained two regression models - one to identify factors that influence the usefulness of CR comments and the other to identify factors that improve the odds of 'Functional' defect identification over the others.

Key findings: The results of our study suggest that a CR comment's usefulness is dictated not only by its technical contributions, such as defect findings or quality improvement tips but also by its linguistic characteristics, such as comprehensibility and politeness. While a reviewer's coding experience positively associates with CR usefulness, the number of mutual reviews, comment volume in a file, the total number of lines added /modified, and CR interval have the opposite associations. While authorship and reviewership experiences for the files under review have been the most popular attributes for reviewer recommendation systems, we do not find any significant association of those attributes with CR usefulness.

Conclusion: We recommend discouraging frequent code review associations between two individuals as such associations may decrease CR usefulness. We also recommend authoring CR comments in a constructive and empathetic tone. As several of our results deviate from prior studies, we also recommend more investigations to identify context-specific attributes to build reviewer recommendation models.

Keywords code review · usefulness · productivity · empirical

1 Introduction

Code Review (CR) is a software development practice where developers asynchronously inspect peers' code changes to identify defects as well as potential quality improvement opportunities [3]. By addressing the cost ineffectiveness of traditional Fagan inspections [20], CR has achieved widespread adoption among commercial and Open Source Software (OSS) development organizations [3, 64, 67]. Many projects mandate each code change be approved through a CR process before it can be integrated into the project's main codebase [65]. As a result, both commercial and OSS developers are spending 10-15% of their time on CR tasks [11]. Due to the large efforts associated with CRs, even a minor improvement in CR's effectiveness can save developers time and can incur significant savings for a software development organization. Since prior studies report between 20 to 44% of CRs are marked as 'not useful' by

A. Turzo, and A. Bosu
Department of Computer Science, Wayne State University, Detroit, Michigan.
E-mail: asifkamal@wayne.edu, amiangshu.bosu@wayne.edu

code authors [13,32,63], improving CR effectiveness is a high priority for many organizations [13,32].

The basic building blocks of a CR are suggestions (aka review comments) authored by the reviewers. A CR is effective if comments belonging to that CR are useful to the code author and involved reviewers [13]. Therefore, an empirical understanding of what makes CR comments useful to code authors and reviewers is essential to improve CR effectiveness. Among the various approaches to improve CR effectiveness, which include adding features to CR tools [6,36,71], automated reviews [4,77,78], reviewer recommendation [76,82], and promoting useful feedback [13,45], this study focuses on the latter. Findings of recent studies investigating CR usefulness suggest that although a developer's primary expectations from a CR comment are the identification of defects, code improvement opportunities, or an alternative solution approach [3,11], the majority of the CR do not meet those expectations [3,9,13,17]. A CR comment may still be considered useful, even though it did not meet any of the primary expectations, but satisfies other secondary criteria, such as improving code maintainability, facilitating knowledge sharing, or helping relationship formed between the participants [3,11]. Although developers have limited disagreements regarding the primary criteria to judge a CR's usefulness, the same cannot be said for those secondary criteria. For example, some developers consider suggestions to improve code documentation or code visualization as useful, while others consider the opposite [32]. Therefore, questions remain about how those secondary criteria compare to each other among a broader pool of developers.

We have identified three knowledge improvement areas based on existing works investigating CR usefulness [3, 13, 30, 46, 67]. First, prior studies on CR usefulness mostly focused on commercial organizations [3, 13,32,46,63,67], except Kononenko *et* al., which investigated CR usefulness among Mozilla developers [45]. Therefore, we have limited insights regarding OSS developers' perspectives on useful CRs. Second, we have a limited understanding of how various contextual and participant factors may influence CR usefulness. Although Bosu *et* al. analyzed how several changeset and reviewer characteristics influence CR usefulness [13], several crucial factors (i.e., detailed in Table 6) were missing from their investigation. These missing insights can be helpful in training reviewers, preparing code changes for CRs, and selecting the best reviewer for those changes. Finally, to analyze CR usefulness factors, Bosu *et* al. [13] grouped code reviews into two categories, either 'Useful' or 'Not useful'. However, such a coarse grain analysis fails to identify crucial insights since a CR suggesting a 'variable renaming' falls into the same bin as the one finding a 'critical defect'. Recommendations obtained through this coarse-grained analysis are prone to amplifying trivial issues such as naming conventions and documentation since the majority of CR comments belong to that categories [9,13,32,58]. As current recommendations on conducting CRs, as well as code reviewer recommendation systems, are designed to repeat past histories rather than to maximize bug finding [26, 62, 82], a large number of functional defects escape CRs [17,59].

To fill in these three knowledge gaps, the primary objective of this study is *to develop a finer grain understanding of what makes a code review comment useful to OSS developers, to what extent a code review comment is considered useful to them, and how various contextual and participant-related factors influence its degree of usefulness.* On this objective, we designed a mixed-method three-step case study of the OpenDev community (formerly known as OpenStack). In the first step, we randomly selected 2,500 code review comments from the OpenDev Nova project and manually categorized those comments to identify the frequency of different categories of review comments. In the second step, we designed an online survey using samples from the first step to better understand developers' views regarding the usefulness of various categories of CR comments. We distributed the survey among the OpenDev developers and obtained 160 usable responses. In the third step, we combined the insights obtained from the survey with the manually categorized dataset to develop two regression models to identify underlying factors that are associated with CR usefulness as well as the types of CR comments being authored. This study's design is motivated by Bosu *et* al., as this study has overlapping objectives with theirs [13]. However, this study's final two steps, as well as our analysis approach, significantly differ from them.

Primary contributions of this study include:

- A finer-grained understanding from OSS developers' points of view regarding what makes a code review useful;
- An empirically developed ranking of code review comment categories based on perceived usefulness;
- An empirical evaluation of how various contextual and participant-related factors are associated with CR usefulness.
- An empirical evaluation of the factors influencing receiving 'functional defect finding' comments vs not receiving such ones.
- Recommendations for practitioners to improve the code review process.

- To promote replication, our analysis scripts and aggregated dataset are publicly available at https://github.com/WSU-SEAL/CR-usefulness-EMSE.

The remainder of the paper is organized as the following. Section 2 provides a brief overview of the research context. Section 3 introduces our two research questions based on our primary objective. Section 4 details our research methodology. Section 5 and 6 present the results of our first and second research questions, respectively. Section 7 and 8 discuss the key implications of our findings and related works, respectively. Section 9 and Section 10 discuss threats to validity and concludes the paper, respectively.

2 Background

This section presents a brief overview of the contemporary CR process and regression analysis concepts essential to understanding our research method.

2.1 Code Review

Code review is the process of reviewing and rewriting a piece of code iteratively before merging it into the main codebase. Contemporary code reviews are lightweight, asynchronous, tool-based, and time efficient. Popular CR tools include Gerrit, Phabricator, ReviewBoard, GitHub pull request, Critique, and CodeFlow. While these tools vary based on the set of supported features, the primary workflow is tool agnostic.

To start a CR process, a code author uploads code to a repository and creates a review request with a description detailing the CR's goal. The author can invite reviewer(s). Additional contributors with required access can self-assign to review as well. CR tools support reviewers' comprehension through various features, such as a side-by-side view highlighting the differences between the current version and the previous one, the commit history of the file, and potential conflicts. Reviewers can raise concerns regarding a particular code segment by adding inline comments (e.g., Figure 2). The developer (author of the patch) can view the review comments and interact with the reviewers using the same interface. The author can address the comment by modifying the files, uploading a new patch set, or adding responses. If modified, the author asks for a re-review. The reviewer checks the modified files and may approve the change if he/she is satisfied. If not, the reviewer may ask for further changes. This process may repeat for several iterations until the reviewer(s) approves the change or is abandoned.

2.2 Regression analysis

Regression analysis is a powerful statistical approach that helps to analyze how one or multiple independent variables influence one dependent variable [21]. There are two primary uses of regression– (i) predictive analysis and (ii) inferential analysis [1]. In predictive analysis, the goal is to develop a formula to predict the value of a dependent variable based on the values of one or more independent variables. In comparison, an inferential analysis aims to determine whether a particular independent variable has any impact on the dependent variable and the magnitude of that impact if it exists. The inferential analysis differs from the predictive analysis preliminary based on two key factors. First, *multicollinearity*: if two or more independent variables are highly correlated to each other, considering all of those correlated variables together can produce an over-fitting problem in inferential analysis. However, in predictive analysis, multicollinearity is not an issue. Second, R^2: R^2 represents the goodness of fit of a regression model [35]. Higher R^2 is important; however, it is more important in predictive analysis. In an inferential analysis, even with a small R^2, the regression model can provide useful insights regarding the relationships between the independent and the dependent variables [1]. We have used inferential regression analysis in this work.

3 Research Questions

The following subsections introduce the two research questions guiding this research.

3.1 RQ1: What makes a code review useful to OSS developers?

These insights may help OSS participants practice code reviews to improve CR effectiveness. However, 'usefulness' as a complex phenomenon may have multiple associated facets. We divide this question into two subquestions to investigate technical and non-technical facets of CR usefulness. Our first sub-question (i.e., RQ1.A) aims to find out the open-ended opinions of OSS developers about CR usefulness to identify whether the CR usefulness criteria for OSS developers are similar to the ones considered by industry participants [3, 11, 13]. More specifically, we investigate how primary (e.g., identification of defects, code improvement opportunities, or alternative solution approaches) and secondary CR usefulness criteria (e.g., improving code maintainability, facilitating knowledge sharing, or helping relationship formation among teammates.) reported by prior studies rank among OSS participants.

RQ1.A: What are the open-ended opinions of OSS developers regarding code review usefulness?

As we aim to rank CR comments based on the degree of usefulness, some of the secondary CR usefulness criteria, such as relationship formation and knowledge sharing, are difficult to rate by an independent evaluator within the limited context and timeframe of a user study. Therefore, we narrow down our focus into a facet that a study participant can independently determine, i.e., the type of comment authored in a CR. Hence, we ask:

RQ1.B: How do OSS developers rank various categories of code review comments based on their degrees of usefulness?

3.2 RQ2: Which contextual and participant characteristics are associated with CR usefulness?

We term attributes of a CR context (i.e., where a CR occurred) as contextual factors. For example, characteristics of changes under review (e.g., code churn and number of files) belong to contextual factors. On the other hand, attributes of the CR participants (e.g., author's /reviewers' experience and prior history with the files under review) belong to participant characteristics. We focus only on contextual and participant-related factors since we aim to identify actionable recommendations for review preparation and reviewer selection to maximize the likelihood of receiving useful feedback. Table 6 provides a list of contextual and participant factors selected based on prior studies. We divide this question into two sub-questions. Our first sub-question aims to identify how various contextual and participant factors may contribute to maximizing CR usefulness.

RQ2.A: How are various contextual and participant characteristics associated with the degree of usefulness achieved in a code review?

Our second sub-question aims to maximize primary CR usefulness criteria, such as identifying functional defects. Therefore, we investigate how various factors are associated with these primary usefulness criteria instead of secondary ones.

RQ2.B: How are various contextual and participant factors associated with identifying functional defects?

4 Research Methodology

This section details our project selection, data collection, survey design, and qualitative and quantitative analyses approach.

4.1 Project selection

We selected the OpenDev community for the survey and selected the OpenDev Nova project for our quantitative analysis for the following reasons:

- OpenDev is one of the largest OSS development communities with more than 15K contributors, and OpenDev developers practice tool-based CRs [11].
- The OpenDev family includes 52 different projects as of 2023 [28], which includes Nova, Neutron, Heat, and Swift.
- OpenDev community includes approximately 110k persons from more than 700 organizations spanning 182 countries worldwide [22].
- OpenDev Nova is one of the most active projects in the OpenDev community [39], and it has been the subject of prior studies investigating CRs [30,81].

4.2 Data Mining

The CR repository of the OpenDev community is managed by Gerrit[1]. We use a Java-based Gerrit Miner tool to access Gerrits' REST API to mine all the publicly available CRs from the OpenDev's repository[2] and store those in a MySQL database. Our dataset includes a total of 795,226 completed (i.e., either 'Merged' or 'Abandoned') CRs spanning July 2011 to March 2022.

4.3 Classification Rubric

We have identified five prior studies that have presented classifications of issues identified during CRs [9,13,32,49,58]. We started with the classification proposed by Beller et al. [9]. In the proposed classification scheme, Beller *et* al. classified CR changes into two higher-level categories: 'Functional' and 'Evolvability'. They divided 'Functional' and 'Evolvability' into six and three sub-categories, respectively. Two of the 'Evolvability' sub-categories ('Structure', and 'Documentation'), were further divided to form two lower-level sub-categories from each.

[1] https://www.gerritcodereview.com/
[2] https://review.opendev.org

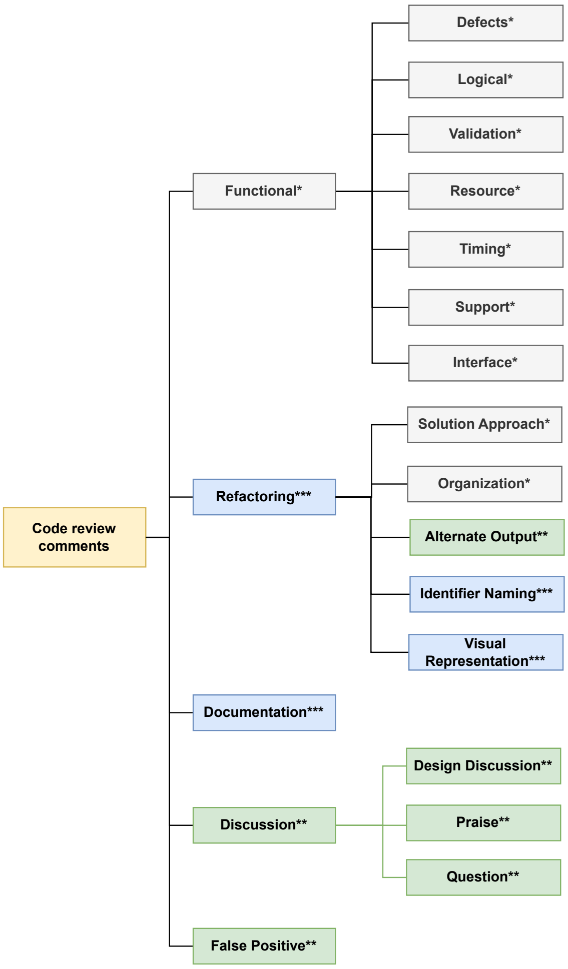


Fig. 1: Adopted code review comment classification scheme. * → components adopted directly from Beller *et* al., ** → newly added categories, and *** → modification of Beller *et* al. scheme.

While we use their scheme as the basis for our proposed one, we made the following three modifications to fit our study context better.

1. Beller *et* al.'s scheme is targeted towards CR changes [9], whereas we focus on CR comments. Prior studies have found a non-trivial number of CR comments that do not trigger code changes [13, 32]. For example, Bosu *et* al. found four such categories: i) 'Praise', ii) 'Questions', and iii) 'Design discussion' [13]. Later Hasan *et* al. also found those categories of CR comments [13]. Therefore, we introduce a new category named 'Discussion' that includes these three sub-categories.
2. A 'False positive' is a CR comment where the reviewer's concern is invalid. Beller *et* al. excluded false positives [9], since those do not induce changes. We include a new category named 'False positive' to include those, as we target CR comments instead of changes.
3. Instead of four levels in Beller *et* al.'s schema [9], we simply our schema into three levels by excluding the 'Evolvability' category and putting its subcategories directly under the top level.
4. We rename the 'Structure' subcategory from Beller *et* al. into refactoring, since types of changes belonging to fall under Martin Fowler's refactoring catalog [23]. We add the 'Alternate output', another sub-category discovered by Bosu et al. [13] under this group as well. Finally, we also include 'Visual representation' under 'Refactoring', as those were included among refactorings by prior studies [33, 80].

Figure 1 shows our CR classification scheme. Components in one star (*) are taken from Beller *et* al [9] without modification. Components in two stars (**) and three stars (***) are our additions and modifications, respectively. Table 1 provides a brief description of each of the CR comment categories from this scheme.

4.4 Manual Labeling

For this phase, we randomly selected 2,500 CR comments from the OpenStack Nova project. We chose OpenDev Nova has the highest number of distinct contributors as well as the highest number of completed CRs within the OpenDev community. According to an investigation of our dataset, during the period of July 2011 to March 2022, Nova has 38,775 either completed or abandoned CRs. The second highest is the Neutron, which has 23,456 either completed or abandoned CRs during the above-mentioned period. Our collected dataset of the Nova project contains a total of 300,304 code review comments from July 2011 to March 2022. We selected 2,500 as the sample size to achieve a 95% confidence interval with a 2% margin of error [79]. Two of the authors independently categorized each of the selected comments into one of the eighteen categories from Table 1 after reading the entire discussion thread associated with a comment and also inspecting its associated code context. Each author also labeled each comment as either 'useful' or 'not useful'. For this labeling, we adopt the 'usefulness' labeling rubric proposed by Bosu *et* al. [13]. Two subsequent CR usefulness studies [30, 63] have also used this rubric, which is as follows:

– *Useful*– A code review comment is considered 'useful':

Table 1: Description of the categories from our code review comment classification scheme

Group	Category	Description
Functional	Functional defect	Functionality is missing or implemented incorrectly and such defects often require additional code or larger modifications to the existing solution.
	Logical	Control flow, comparison related, and logical errors.
	Validation	Validation mistakes or mistakes made when detecting an invalid value are of this class. Any kind of user data sanitization-related comments are in this category, too.
	Resource	Resource (variables, memory, files, database) initialization, manipulation, and release.
	Timing	Potential issues due to incorrect thread synchronization.
	Support	Issues related to support systems and libraries or their configurations.
	Interface	Mistakes when interacting with other parts of the software such as - existing code library, hardware device, database or operating system.
Refactoring	Solution approach	Suggestions to adopt an alternate algorithm or data structure.
	Alternate Output	Comments that suggest modifying the error message, toast message, alert, or change what is returned by a function.
	Organization of the code	Refactoring suggestions such as those included in Martin Fowlers's catalog [23].
	Naming Convention	Violations of identifier naming conventions.
	Visual Representation	Whitespace, blank lines, code rearrangements, and indentation-related comments.
Documentation	Documentation	Suggestions to add /modify comments or documentation to aid code comprehension.
Discussion	Design discussion	Discussions on design direction, design pattern, and software architecture.
	Question	Questions to understand the design or implementation choices.
	Praise	Complement for a code.
False positive	False positive	If a review comment raises an invalid bug or concern.
Others	Other	Comments not belonging to any of the above categories.

1. if the code author explicitly acknowledges the identified issue (e.g., "Good catch").
2. If the code author implicitly acknowledges the identified issue by making the suggested changes.
3. If the author explicitly defers the identified issue to a future change.

- *Not useful*– A comment is considered 'not useful':
  1. if the code author explicitly states the issue as invalid.
  2. if the code author neither makes the suggested changes nor acknowledges deferring it to a future change.

We computed the level of inter-rater reliability of this multi-label manual categorization process (assigning comments into 18 categories) using Cohen's kappa (κ) [16] which was measured as 0.68 ('a substantial agreement'[3]). Cohen's kappa value for the 'useful'/'not useful' categorization is 0.84.

4.5 Survey Design for RQ1

To better understand what makes a CR useful to OSS developers (RQ1), we designed an online survey for persons who have first-hand knowledge of participating OSS CRs (i.e., actively participated in CRs by providing suggestions).

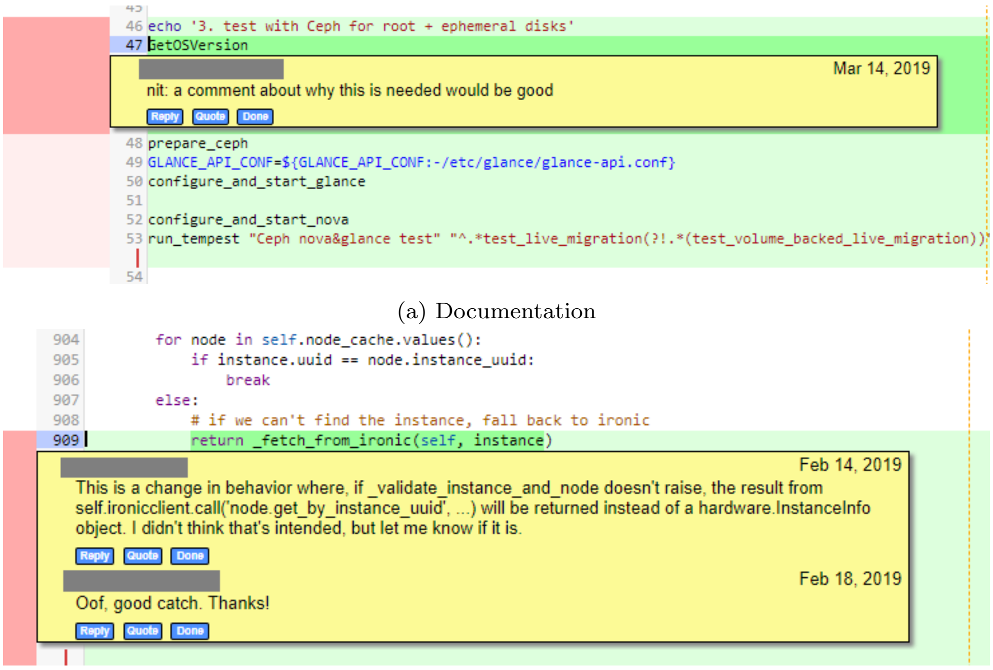
4.5.1 Questionnaire preparation

Since RQ1.A aims to find out the open-ended opinions of OSS developers about CR usefulness. Our survey asks, 'What makes a code review useful (Q5 in Table 2)?'. RQ1.B aims to understand how useful each kind of code review comment listed in Table 1 is. On this goal, we include two categories of questions in the survey. First, we ask the respondents to manually rate each CR category on a five-point Likert scale, only based on a short definition of that category (Q7). Second, we ask the respondents to rate sample CR comments on a 10-point scale, by only showing screenshots and without revealing the actual categories those comments belong to (Q8 to Q39), in total, there were 32 CR comment samples. To identify sample CR comments for this 10-point scale rating, we identified multiple ideal

[3] Kappa (κ) scores are commonly interpreted as: 0.01–0.20 as 'none to slight,' 0.21–0.40 as 'fair,' 0.41– 0.60 as 'moderate,' 0.61–0.80 as 'substantial,' and 0.81–1.00 as 'almost perfect agreement' [47].

Table 2: Questions included in our survey

#	RQ#	Question Text	Answer Choices
Q1	D	What is your highest level of education?	[High school, Bachelors, Masters, Ph.D., Other]
Q2	D	How many years of software development experience do you have?	[Numeric input]
Q3	D	How many years have you been practicing tool-based code reviews?	[Numeric input]
Q4	D	How many hours per week, on average, do you spend reviewing other contributors' code?	[Numeric input]
Q5	RQ1	In your opinion, what makes a code review useful?	[Free form text]
Q6	D	Approximately, what proportion of code reviews in your project, do you feel are useful (i.e., fits the above definition)?	[Numerical slider from 0%-100%]
Q7	RQ1	Rate the following categories of review comments based on their perceived usefulness to you. Choices are 16 categories from Table 1 excluding the 'False positive' and 'Others' categories.	Matrix table [61] with [1-5 Star Rating]
Q8-Q39	RQ1	How useful do you find the above code review on a scale of 1 to 10 if you were the author?	Each question showed a screenshot such as Figure 2 [1-10 Star Rating]



```
echo '3. test with Ceph for root + ephemeral disks'
GetOSVersion
```

[redacted] Mar 14, 2019

nit: a comment about why this is needed would be good

```
prepare_ceph
GLANCE_API_CONF=${GLANCE_API_CONF:-/etc/glance/glance-api.conf}
configure_and_start_glance

configure_and_start_nova
run_tempest "Ceph nova&glance test" "^.*test_live_migration(?!.*(test_volume_backed_live_migration))
```

(a) Documentation

```
        for node in self.node_cache.values():
            if instance.uuid == node.instance_uuid:
                break
        else:
            # if we can't find the instance, fall back to ironic
            return _fetch_from_ironic(self, instance)
```

[redacted] Feb 14, 2019

This is a change in behavior where, if _validate_instance_and_node doesn't raise, the result from self.ironicclient.call('node.get_by_instance_uuid', ...) will be returned instead of a hardware.InstanceInfo object. I didn't think that's intended, but let me know if it is.

[redacted] Feb 18, 2019

Oof, good catch. Thanks!

(b) Validation

Fig. 2: Examples of code review comments included in our survey

examples for each CR comment category from our manually labeled dataset. We consider an example ideal if its category can be distinguished unambiguously by reading the comment and its context (i.e., screenshot). To be an ideal example, a CR comment: i) must clearly state what needs to be changed or what the comment pertains to, and ii) its surrounding code context must include the relevant code snippet.

However, we excluded the 'False positive' category as those are not useful (i.e., 0). We also excluded 'Others,' as we could not find a representative sample for that category. With the remaining 16 CR comment categories, two of the authors independently investigated our labeled dataset. They identified a total of 64 candidate examples, where exactly four examples belonged to each CR comment category. We merged these two lists, assigning a score of '2' to examples belonging to both lists and assigning '1' to others. Next, we had a discussion section to pick two of the most representative examples for each CR comment category, where examples with a score =2, got higher priorities. At the end of this process, we selected a total of 32 examples. We took Gerrit screenshots of the selected examples with surrounding code contexts. Figure 2 shows two of the examples included in our survey. Table 2 lists the questions included in our survey. We have also included all the questions and 32 selected examples in our supplementary material.

4.5.2 Pilot Survey

For expert reviews, we sent the survey questionnaire to two renowned Software Engineering (SE) researchers with expertise in code reviews. According to their feedback, we made several edits for structural and linguistic modifications. Finally, we piloted the survey with three SE graduate students for validation. The goal of piloting the survey is to identify any visible mistakes and to get any modification suggestions for improving the survey questions. We got the final survey questions, consent form, participant selection strategy, solicitation email, and data management protocol reviewed and approved by our university's Institutional Review Board (IRB).

4.6 Data Collection

To solicit the opinions of participants with adequate CR experience, we sent our survey only to developers who have submitted at least 5 code changes for review. Since all the code snippets are written in Python, we also limited our selection to OpenDev projects with Python listed as the primary language. We identified a total of 4,188 OpenDev developers satisfying these criteria.

We sent personalized emails, composed according to the guidelines provided by our IRB, to each of the 4,188 developers with the link to the survey hosted on Qualtrics [70]. Since we obtained the email addresses by mining Gerrit, there are ethical considerations [5] for sending such invitations. Therefore, each of our invitations: i) indicated the OSS project that we mined to obtain his/her email address, ii) indicated if the participant would get any more emails from us, iii) provided a link to opt-out, and iv) apologized for the inconvenience. Excluding the 1,010 undelivered ones (i.e., email address no longer valid), we consider 3,178 emails as delivered. We obtained a total of 238 responses, a response rate of 7.49%.

4.7 Survey data analysis

We checked the quality of the recorded responses and excluded 78 cases where the respondent did not answer at least one survey question that is tied to our research questions (i.e., left the survey after completing only the demographic section). So, we conducted our analysis on 160 responses. For the one qualitative question (i.e., Q5 in Table 2), we adopted a systematic data analysis process. Two of the authors independently went through all the responses to extract the factors of useful CRs mentioned by our respondents. Then we had a discussion session to cross-validate and create a finalized list of factors. With this list, two of the authors independently labeled each response to one or more codes. After completion, we compared the labels to identify conflicts. Finally, we had a discussion session to resolve those conflicting labels. Inter-rater reliability based on Cohen's kappa κ for this labeling was 0.56 (i.e., 'moderate'). The low κ value is because a single response might mention one or multiple CR usefulness criteria, which creates theoretically exponential possibilities.

Table 3 provides an overview of the demographics of our survey respondents. Our respondents include mostly experienced software developers, with 79% having more than five years of software development experience. The respondents also have significant code review experiences, as almost 90% have been practicing tool-based code reviews for more than two years. On average, our respondents spend 5 hours per week in code review, which is on par with the numbers reported in prior studies [11, 32]. Based on these characteristics, we believe that our respondents are eligible to provide valuable opinions to answer our research questions.

5 Results: (RQ1) What makes a code review useful to OSS developers?

The following subsections describe the results of our first research question based on our survey responses.

5.1 RQ1.A: What are the open-ended opinions of OSS developers regarding code review usefulness?

Based on our respondents' opinions, we identified 23 different criteria (codes) to judge a CR as useful. Posthoc, we aggregated those codes into 9 higher-level groups. Table 4 shows those nine groups, 23 codes, and the percentage of respondents mentioning the code. The sum of the percentages exceeds 100% since respondents could mention multiple codes in their responses. Unsurprisingly, 'Finds defects' tops this list, followed by 'Improve code quality'. However, we notice a strong emphasis on non-technical aspects, with one-third of respondents judging CR's usefulness based on linguistic characteristics. We also ran Chi-Square (χ^2) tests to check whether these criteria differed based on education level, software development experience, or code review experience. We found significant differences ($\chi^2 = 13.34, p = 0.012$)[4]

[4] $p - values$ are adjusted using Benjamini and Hochberg corrections [10] due to multiple comparisons.

Table 3: Respondents' demographics

Question	**Mean**	**Median**	**Categorical description**	**% of respondents**
Q1. What is your highest level of education?	—	—	High school	36.6%
			Bachelors	50.7%
			Masters	5.4%
			Ph.D.	4.4%
			Other	2.9%
Q2. How many years of software development experience do you have?	10.14 years	10 years	< 2 years	1.5%
			2-5 years	19.5%
			6-10 years	33.7%
			> 10 years	45.3%
Q3. How many years have you been practicing tool-based code reviews?	5.43 years	5 years	< 2 years	10.2%
			2-5 years	44.9%
			6-10 years	28.8%
			> 10 years	16.1%
Q4. How many hours per week, on average, do you spend reviewing other contributors' code?	5.10 hours	4 hours	< 2 hours	23.9%
			2-5 hours	42.9%
			6-10 hours	15.1%
			> 10 hours	18.1%

Table 4: Codes that emerged from open-ended survey question and category that we assigned each code to

Assigned category (% respondents)	**Theme (# of respondents)**
Find defects (44.4%)	Defect finding (126)
	Security issues (11)
	Fulfill requirements (11)
	NIT/typos (9)
Improve code quality (42.5%)	Code optimization (48)
	Alternative implementation (35)
	Improve documentation (21)
Uses appropriate language (28.1%)	Constructive criticism (29)
	Respectful (15)
	Concrete example (13)
	Understandable (12)
Maintainability (28.1%)	Code comprehensibility (23)
	Maintain design (17)
	Reduce maintenance cost (6)
Facilitates knowledge sharing (25.0%)	Knowledge transfer (35)
	Mentoring (3)
	Project awareness (10)
Facilitates better design (24.4%)	Design suggestion (27)
	Consider wider context (13)
	Questions leading to better design (12)
Appreciate good work (3.1%)	Appreciate good work (5)
Helps community building (3.1%)	Helps community building (5)
Timely feedback (1.8%)	Timely feedback (3)

only based on software development experience as veteran developers with more than ten years of experience were more likely to use the 'finds defect' criteria.

The following subsections detail those nine CR usefulness criteria based on our respondents' opinions. We also include verbatim excerpts to illustrate their opinions further. Each excerpt is associated with a numeric value that indicates a unique identifier for the respondent who expressed that opinion. For example, [#40] indicates a response from the 40^{th} respondent.

5.1.1 Finds defects

Most of the respondents (44.4%) consider a code review as useful if it points out a defect. Identification of any functional defects (i.e., the functional group from Table 1) is useful and is one of the primary expectations of code authors [3].

> *Points to errors or issues, that the author didn't see during writing code. [#40]*

Prior research found evidence of security defects identified during code reviews [12, 59]. Although the identification of security defects is rare, they are extremely useful.

Any code review that aims to improve the code is useful. Sometimes the improvement is drastic and critical, such as pointing out an error that would lead to a crash or a security issue. [#116]

Code reviews can also serve as a validation step as reviewers may provide feedback on requirement fulfillment.

Proper scoping of the reviews:- Architectural reviews, alignment with requirements ... [#52]

5.1.2 Improves code quality

Any suggestions to improve code quality are useful. A large number of our respondents appreciate suggestions to achieve better quality through an alternate solution approach.

Clear, constructive discussion of the code, ..., suggestions from reviewers for alternative implementations if there are areas the code could be improved or does not meet criteria. [#103]

Code optimization is another quality improvement area, as reviewers might suggest possible code reuse opportunities and code optimization techniques.

Optimization of code e.g. newcomers may not know full design so they may implement an already existing function which could be found in code review. [#206]

Documentations to aid program comprehension are crucial for long-term maintenance. Pointing out mistakes in documentation or documentation improvement suggestions is useful.

Finding structural issues/code comments /documentation that would make the code hard to maintain going forward. [#130]

5.1.3 Uses appropriate language

To be useful, a CR comment needs to be authored in an appropriate language as one of our respondents mentioned,

.. it's not only about what was written but also how it was written. [#56]

Harsh critiques may demotivate an author and are less likely to be useful. Moreover, reviews must be concise, understandable, and explanatory so that authors can easily understand their mistakes and make necessary changes.

I typically write code reviews in the form of 'appreciate - suggest a change or point out the mistake - appreciate again' This seems really useful to me because it does not undermine the developer's confidence and at the same time tells them about how to go about enhancing or correcting their approach. [#221]

Sometimes examples illustrating potential problems are extremely useful to help authors understand problems with their proposed solutions.

...reviewers must clearly state their concerns and opinions and where applicable point to specific code lines, documentation, mailing list threads, or other discussions that are relevant to the matter at hand or provide examples. [#10]

Even if a reviewer rejects a change, it is important to do so in a constructive way.

If negative, provides constructive feedback, the more definitive the better. [#101]

5.1.4 Improves 'maintainability'

Code reviews checking for violations of project design constraints are also useful

1. what are the problems about the code? 2. Does the code align with the project's blueprint? [#141]

Code that is not readable is difficult to maintain. Future contributors will face difficulty to modify the less readable code. Suggestions to improve readability from a different person's perspective are useful.

...pointing and discussing better ways of achieving the same result. by better ways I mean: code readability, performance, and maintainability. [#215]

5.1.5 Facilitates knowledge sharing

Prior studies suggest knowledge sharing as one of the most important benefits of code reviews [3, 11]. Code review is a bidirectional knowledge transfer process, the code owner learns from the reviewers' comments and the reviewer also learns from the code owner. Reviews facilitating such knowledge transfer are useful.

Knowledge transfer. It lets other members of the community share their experiences with others. Sometimes the more experienced review participant knows a better way to do things. Sometimes the more experienced review participant is aware of past design decisions or design constraints the other party is not aware of. [#8]

A useful review facilitates the newcomers to learn code architecture, team culture, and coding conventions.

Friendly relations with the core developers, other participating developers, and mentorship. [#166]

5.1.6 Facilitates a better design

Reviewers provide useful suggestions on design direction and design patterns which makes software products better architectured.

> *...lead to a discussion about potential better solutions (for bug fixes) or future design (for implementing new features). [#121]*

Code review is the platform for reviewers to identify errors and for the code owner to defend their work or make it correct. Such discussions help increase the breadth of knowledge of both reviewers and code owners.

> *Providing the wider context (e.g. interactions with modules or systems the code author did not consider). [#1]*

Asking critical questions about software architecture leads to a better design of the product. Moreover, when the reviewer asks questions about some change, that means the code is not sufficiently self-explanatory. So from the readability perspective, there is an opportunity for improvement.

> *Highlighting issues in the code or asking relevant questions to the topic which could lead to improvements. [#111]*

5.1.7 Appreciates good work

Positive encouragement is essential for inspiring new contributors to do quality work and increasing their confidence. Many respondents find such reviews useful.

> *... code contributor gets more confident in the code that has been appreciated and is encouraged to continue contributing. [#41]*

5.1.8 Timely feedback

As review delays can frustrate authors, useful feedback should be timely and not create confusion.

> *...Ideally, all concerns should be expressed at once, avoiding back and forth communication between the reviewer and the code author, which can significantly delay the patch approval. [#88]*

5.1.9 Helps community building

Constructive criticism creates openness and better programming habits within a community and therefore is useful.

> *Build team culture for better programming habits, confidence in the artifacts, openness in communication. [#6]*

Key takeaways: *According to OpenDev developers, a CR's usefulness is dictated not only by its technical contributions such as defect finding or quality improvement tips but also by linguistic and process aspects such as comprehensibility, tone, and timeliness.*

5.2 RQ1.B: How do OSS developers rank various categories of code review comments based on their degrees of usefulness?

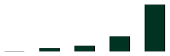
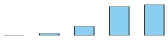
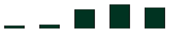
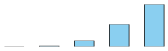
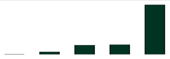
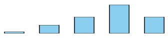
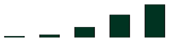
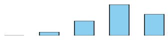
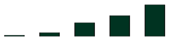
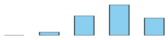
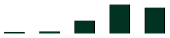
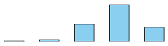
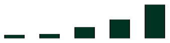
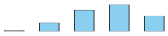
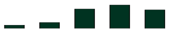
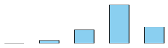
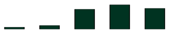
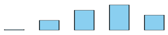
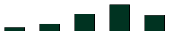
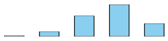
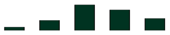
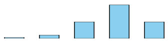
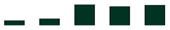
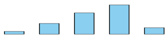
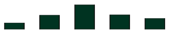
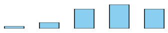
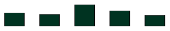
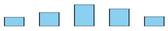
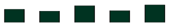
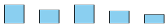
In response to Q7 in Table 2, our respondents rated 16 CR comment categories on a five-point scale. The three columns under the *User perception* header in Table 5 show the distributions of the ratings provided by our respondents, the average rating for each comment category, and the standard deviation of the assigned ratings.

Q8-Q39 asked the respondents to rate 32 selected example code reviews on a ten-point scale based on their usefulness (i.e., how useful they find a code review on a scale of 1 to 10 if they were the author). Since we included two examples for each category, a respondent's *Sampled user rating* for a comment category was computed by taking the average of their two scores for the examples of that particular category. For comparison against the *User perception* score of a category, we divided the average *Sampled user rating* by 2. The three columns under the *Sampled user rating* header in Table 5 show the distributions of the ratings assigned to sample code reviews from each category. The *Ratio of comments* column shows the percentage of review comments belonging to a particular category in our manually labeled dataset of 2,500 comments. Finally, the *Avg. Rating* for the category was computed by taking the average of *User perception* and *Sampled user rating* for that category. The rows in Table 5 are sorted based on this score.

Our results suggest that finding functional defects, validation, and logical issues are top priorities for our respondents. These results support findings from prior studies [3, 11]. By comparing this ranking against the classification scheme presented in Table 1, we find most of the categories belonging to the 'Functional' group ranking at the top. However, only 19% of CR comments from our dataset belonged to these categories. This ratio is similar to the ones reported in prior studies [9, 13, 32]. Although 'Documentation' and 'Organization of code' rank among the bottom half (i.e., 9th and 10th respectively), more than 40% of CR comments belonged to those categories. These results also support the prior observation that most CR comments are related to trivial issues [17].

We also noticed a few discrepancies between the ranking based on avg. *User perception* and the one based on avg. *Sampled user rating*. While avg. *User perception* ranking for the 'Validation' category places it eighth in the list, and below the 'Documentation' category, it ranks top according to avg. *Sampled user rating*. On the contrary, our respondents rated logical cat-

Table 5: Ranking of code review comments based on usefulness rating

Comment category	**User perception**			**Sampled user rating**			**Ratio of comments**	**Avg. Rating**
	Distribution	**Avg (μ).**	**SD (σ)**	**Distribution**	**Avg. (μ)**	**SD (σ)**		
Functional Defect		4.50	0.82		4.26	0.89	0.48%	4.38
Validation		3.81	1.05		4.50	0.73	3.68%	4.16
Logical		4.51	0.84		3.71	1.16	2.28%	4.11
Interface		4.19	0.96		4	0.96	1.60%	4.10
Solution Approach		4.12	0.98		3.88	0.92	8.48%	4.00
Question		4.08	0.91		3.89	0.93	13.56%	3.99
Design Discussion		4.07	1.13		3.66	1.21	3.64%	3.87
Resource		3.69	1.09		3.96	1.20	1.48%	3.83
Documentation		3.83	1.0		3.63	1.17	33.32%	3.73
Organization of Code		3.62	1.10		3.74	1.0	7.68%	3.68
Alternate Output		3.39	1.06		3.87	0.93	2.56%	3.63
Timing		3.62	1.18		3.37	1.12	0.16%	3.5
Naming Convention		3.11	1.20		3.75	1.10	3.52%	3.43
Praise		2.99	1.30		3.07	1.35	4.20%	3.03
Visual Representation		3.12	1.45		2.71	1.50	3.92%	2.92

egories as the top, but the samples' ratings place that category at tenth. The high standard deviation for the sample ratings suggests that although respondents consider CR identifying logical mistakes as highly useful, some of the respondents rated the logical samples from our survey as less severe issues. The 'Praise' and 'Visual representation' categories ranked among the lowest. We also noticed the highest standard deviations (SD) for these two categories. These results suggest that developers have contradictory opinions regarding these categories, as some developers perceive those as useful, while others perceive those as less useful. Other categories where our respondents' opinions have diverging opinions (i.e., high SD) besides these two are 'Resource synchronization', 'Design discussion', and 'Documentation'.

Key takeaways: *While OpenDev developers consider 'Functional' CR comments as most useful, less than 20% CR comments from our sample belonged to that group. OpenDev developers also had widely varied views regarding the usefulness of comments belonging to praise, documentation, design discussion, resource synchronization, and visual representation.*

6 RQ2. Which contextual and participant characteristics are associated with CR usefulness?

To answer this question, we selected a total of 21 attributes based on prior code review studies [13, 25, 32, 51, 63]. A total of 9 attributes are related to review participants and the remaining 12 are related to the review context. For each attribute, Table 6 provides a short definition and a brief rationale on why this attribute may influence the usefulness of CR comments. In a similar investigation, Bosu *et* al. selected four participant factors and two contextual factors. We selected three of the four participant factors from their study, as OSS projects do not have the 'same team' concept as Microsoft. We selected one of the two contextual factors from Bosu *et* al. We do not consider file extension in our analysis, as more than 90% of the sample belongs to one extension (i.e., 'py').

We would also like to mention that we do not include any attributes related to the CR comment text due to our design constraints. Based on the results of RQ1.A, linguistics attributes of a review seem good attributes to be included in our analyses. However, we created our labeled dataset before the survey since we needed samples for the survey design. Our manual labeling rubric did not consider linguistics aspects. Judging non-technical aspects such as 'respectfulness,' 'understandable,' and 'constructive criticism' depends on the participants; therefore, a third-party rater would not have judged those accurately.

We use Python scripts and MySQL queries to compute those attributes for the 2,500 manually labeled CR comments from the OpenDev Nova project. After removing the redundant variables, we use this dataset to develop two regression models, the first one is a Linear Regression Model (LR) to investigate how these factors the degree of usefulness achieved in a CR (RQ2.A), and the second one is a Multinomial Logistic Regression (MLR) model to identify how these factors associate with identification of functional defects. The following subsections detail our analysis approach and results for the two sub-questions for RQ2.

6.1 RQ2.A: How are various contextual and participant characteristics associated with the degree of usefulness achieved in a code review?

To answer this question, we trained a linear regression (LR) model using the `glm` function from the `stats` library in R. In an LR model, if the dependent variable is Y and the independents are $X_1, X_2, X_3, ..., X_n$, then `glm` tries to fit a curve of the form $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + ... + \beta_n X_n$ on the dataset. Here β_0 is the intercept and β_i is the coefficient of the independent variable X_i. The β_i, also indicates the change in the dependent's value if X_i changes by one unit, while all the other independents remain constant. Therefore, a positive β_i indicates a positive association of X_i with Y and vice versa. The following subsections detail our model training, evaluation, and results.

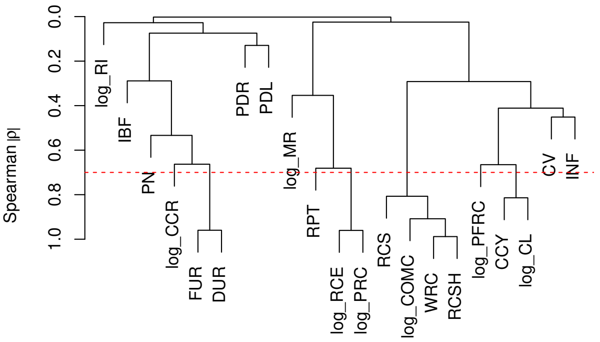


Fig. 3: Hierarchical structure among the independent variables according to Pearson correlation

6.1.1 Model training and evaluation

We noticed highly skewed distributions for eight out of the 21 selected attributes. The list of skewed variables includes COMC, RI, CCR, RCE, MR, PRC, PFRC, and CL. Therefore, we applied log transformations (i.e., log_{10}) to model those variables' associations more accurately with our dependent variables than without such transformations [14, 56].

Multicollinearity problems occur when several variables in a multivariate regression have a high degree of association with one another. The impacts of individual variables may not be accurately detected by a model trained with multicollinear factors [48]. We used Sarle's Variable Clustering (VURCLUS) method to find highly correlated factors [69]. We constructed a hierarchical cluster representation of the independents using Spearman's rank-order correlation test. Based on Hinkle *et* al.'s suggestion [37] we chose the correlation coefficient $|rho| = 0.7$ as the cutoff. Only one explanatory variable from a cluster that contained multiple explanatory variables with $|rho| \geq 0.7$ was selected. After evaluating correlation and redundancy, we chose the most effective set of factors for our model using Jiarpakdee et al.'s [42] `Autospearman` R library.

Figure 3 shows the hierarchical structure of the independent variables based on correlation analysis. We identified four highly correlated clusters. From the (RCS, WRC, RCSH, log_COMC) cluster, `AutoSpearman` keeps

Table 6: Contextual and participant factors associated with code review usefulness

Attributes	Description	Rationale
Participant attributes		
File commit count (COMC)*	The number of times a person has made changes to a file.	With multiple prior changes, a person's awareness of the file grows and therefore odds of useful reviews [13, 63])
Weighted recent commit (WRC)	If a file has a total of n prior commits and the author has made three of the prior n commits (e.g., i,j,k), then: $WRC = \frac{(i+j+k)}{1+2+...+n} = \frac{2(i+j+k)}{n(n+1)}$	Recent committers of a file may have a better understanding of its current design [74].
Mutual reviews (MR)	Number of reviews the particular author and reviewer have performed mutually.	High MR indicates more mutual understanding between the author and reviewer, so the reviewer may understand the strengths and weaknesses of the author and may provide a useful review [11].
Prior file review count (PFRC)*	The number of times a person has reviewed the current file.	With multiple prior reviews, a person's awareness of the file grows and therefore odds of useful reviews [13, 32, 63])
Reviewer code share (RCS)	The ratio of the number of code lines written by the reviewer and the total number of lines in the code.	If the reviewer has a higher contribution in the file, then he may have a better understanding of the context and can provide a useful review [13, 32, 63].
Review experience (PRC)	Number of reviews the current reviewer has performed.	Reviewer review experience may have an association with review quality [32].
Reviewer commit share (RCSH)	Ratio of the number of commits made by the reviewer and the total number of commits for the current file.	If a reviewer makes a large number of commits, then she has a higher codebase understanding and provides useful reviews [74].
Coding experience (RCE)	Number of files the reviewer has submitted in the codebase as an author.	If the reviewer has higher project experience, he may have a better understanding of the entire code base [32, 45].
Reviewer project tenure (RPT)*	Tenure of the reviewer in number of months.	Newcomers of a project are primarily added for knowledge dissemination and are less likely to write useful reviews [13].
Contextual attributes		
Patchset number (PN)	The number of total patches of a review.	Higher PN indicates a review going through more iterations due to useful feedback [13, 32].
Comment volume (CV)	Ratio between the number of comment lines and the total number of lines in the current file.	If a file has a high CV, then the code may become easier to understand, and the possibility of providing useful reviews increases.
Review interval (RI)	The interval between the submission of the file to the review tool and the final review outcome	Time required to evaluate a file has an association with the product quality [32, 73, 74].
Code churn (CCR)	The total number of lines that have changed (added and deleted lines).	If the number of changed lines is higher, there is a higher probability of having defects [53, 54]. Identification of defects is considered useful by authors.
Is bug fix? (IBF)	Whether the current patch is for fixing bugs or not.	Bug fixes are often associated with defect-prone files [74]. Reviewers may have higher opportunities to identify more bugs during reviews for bug fixes.
Is new file? (INF)	Whether the current file is added to the patch for the first time or not.	A new file may contain more issues than a file that went through a review process previously.
Directory under review (DUR)	How many directories have been affected for making current modifications	Higher number of directories are indicators of tangled code changes [18], which are difficult to review.
Cyclomatic complexity (CCY)	The cyclomatic complexity described by McCabe [50]	Code comprehension difficulty increases with complexity, and therefore reduces the likelihood of useful reviews.
File under review (FUR)*	How many files has been affected for making current modification	Larger patches are time-consuming to review. As a result such patches often get cursory reviews.
Current loc (CL)	Number of lines of code currently available in the file	Due to additional required efforts for comprehension, larger files may receive cursory reviews.
Patch description length (PDL)	Number of words to describe the patch	Patch description helps reviewers understand the objective of the change, and may help avoid unnecessary questions.
Readability of patch description (PDR)	Flesch reading ease [43] score of patch description	Patch description should be easy to understand to assist reviewers' comprehension

* - attributes also investigated in Bosu et al.'s study [13].

RCS. From the (log_CL, CCY) cluster CCY survives. From (log_RCE, log_PRC) group, log_RCE survives. Finally, from the (FUR, DUR), FUR survives. After excluding these six variables, we use the remaining 15 variables to construct our regression models. In our analysis, the dependent variable is a continuous variable named *usefulness_score* and the independents are 15 of the surviving factors from Table 6.

For each CR comment from our labeled dataset (there are 2,500 CR comments in the labeled dataset), we compute its *usefulness_score* according to the following formula.

i. If a comment is labeled as 'Not useful', we assign a '0' *usefulness_score*.
ii. If a comment is labeled as 'Useful', we check its assigned category. From the *Sampled user rating* column of Table 5, we take the score obtained for this category and that score is its *usefulness_score*.

While we had two other possible options, i.e., 'User perception' and 'Avg Rating', we found potential issues with those choices for usefulness scores. We could not validate that our respondents indeed fully comprehend a CR comment category based on the brief definitions included in our survey. As 'Sampled user ratings' are obtained based on actual ratings of reviews, those are more accurate than the other two.

We assess the goodness of fit for the trained model using McFadden's R^2, which is 0.026 ($p < 0.001$). We performed a log-likelihood ratio to compare our fitted model with a null model. Our test results suggest ($\chi^2 = 69.9$, $p - value < 0.0001$) significant explanatory power of the model over a null model. We acknowledge that the R^2 value achieved by this model is lower than what we had anticipated. This lower value may be due to two primary reasons. First, in a randomly selected code review sample such as ours, the majority of comments belong to trivial issues such as 'Documentation' and 'Naming convention'. As these comments are more likely to be participant or context agnostic, a model trained using randomly chosen samples is less likely to achieve high explanatory power. Second, approximately 77% CR comments in our dataset are useful ones (i.e., scores higher than zero). As a result, we notice a high intercept value (i.e., 3.02). Instead of a randomly sampled one, if we had a dataset with higher ratios of 'Not useful' comments (i.e., score =0), a model trained with such a dataset would have achieved a higher R^2. Regardless, since our goal is to develop an inferential regression model, not a predictive one, a low R^2 score does not invalidate the obtained insights. Moreover, our model significantly outperforms a null model.

6.1.2 Model analysis

Table 7 reports the co-efficient (β_i) for the surviving variables, 95% confidence interval estimates for the coefficients, and statistical significance of the associations (i.e. p-value). A negative coefficient value indicates if this variable increases, the CR usefulness score decreases and vice versa. Each variable's χ^2 value is the difference between -2 *Log Likelihood* of the full model and a model constructed by dropping that particular variable under discussion. A higher χ^2 value indicates a higher explanatory power added by an independent. Our model's parameter estimates suggest a total of five independent variables having statistically significant associations ($p - value < 0.05$ and presented in bold format). Among those, log_RCE (log of reviewers' coding experience) has the highest level of influence with a positive coefficient. This result suggests that experienced coders, who have significant project knowledge due to a large number of prior code changes, are more likely to provide useful comments. On the contrary, RPT (reviewer's project tenure has a negative association). As this association is contradictory to RCE, we further investigated the reasons. We computed persons' tenures from their first interaction with the project. We found many casual contributors who have long tenures but have made very few commits. Less useful CR comments by these casual contributors contribute to the negative association between CR usefulness and RPT. Contrary to our initial assumptions from Table 6, both log_MR (mutual reviews) and log_RI (review interval) also have significant negative associations. Although 'Quid pro quo' reviews are common [11], negative β for log_MR discourages such practices. Our results from RQ2.B also provide more explanation of this finding. We also notice that delayed reviews do not indicate more useful reviews. If a review takes longer than usual, the reviewer may be overloaded and have done a cursory review to avoid further delays. Finally, we noticed CV (comment volume) lowering CR usefulness. Our investigation found that if a file includes higher ratios of comments, reviewers often include suggestions to improve those with better wording, explanations, and grammar fixes. Since these types of suggestions have lower usefulness scores, CV has a negative association.

Key takeaways: *i) While a reviewer's coding experience positively associates with code review usefulness, their number of mutual reviews with the author, code review interval, comment volume in the file, and selection of casual contributors have opposite associations.*

Table 7: Results of or linear regression model to identify how contextual and participant factors associate with the degree of CR usefulness. β in blue background indicates a significant positive association, and red indicates a significant negative association.

Attribute	Co-efficient (β_i)	95 %Confidence interval for β_i	χ^2	p-value
log_MR	-4.02e-02	[-7.64e-02 , -4.03e-03]	4.745	**0.0295** *
RPT	-5.92e-03	[-1.08e-02 , -1.01e-03]	5.590	**0.0181** *
log_PFRC	1.69e-02	[-5.18e-02 , 8.56e-02]	0.232	0.6301
RCS	-6.37e-03	[-2.72e-02 , 1.44e-02]	0.359	0.5488
log_RCE	1.34e-01	[6.90e-02 , 1.99e-01]	16.287	**0.0001** ***
PN	1.67e-03	[-7.19e-03 , 1.05e-02]	0.136	0.7122
CV	-3.66e-01	[-5.69e-01 , -1.63e-01]	12.471	**0.0004** ***
FUR	-1.30e-03	[-8.39e-03 , 5.79e-03]	0.129	0.7186
log_RI	-4.83e-02	[-7.80e-02 , -1.86e-02]	10.154	**0.0015** **
log_CCR	2.99e-02	[-3.52e-02 , 9.51e-02]	0.812	0.3677
IBF	1.88e-03	[-1.55e-01 , 1.59e-01]	0.001	0.9813
INF	1.53e-01	[-1.09e-01 , 4.15e-01]	1.315	0.2517
CCY	4.20e-05	[-2.08e-04 , 2.92e-04]	0.108	0.7414
PDR	-3.72e-04	[-3.48e-03 , 2.74e-03]	0.055	0.8145
PDL	-5.92e-04	[-1.37e-03 , 1.85e-04]	2.229	0.1356

*** , **, and * represent statistical significance at $p < 0.001$, $p < 0.01$, and $p < 0.05$ respectively.

6.2 RQ2.B: RQ2.B: How are various contextual and participant factors associated with identifying functional defects?

When the dependent variable is nominal with more than two levels, MLR allows modeling the likelihood of a particular outcome occurring given the values for a set of independents. MLR is a Maximum Likelihood Estimator, where the log odds of the dependent variable are captured as a linear combination of the independents. Since our dependent variable has five levels (i.e., category of CR comment), MLR is a right fit for our analyses. MLR also has the following advantages:

- MLR can operate in cases where variables are not normally distributed.
- MLR can be applied when there is no linear relationship between independent and dependent variables [8].
- For MLR analysis, independent variables can be both discrete and interval (continuous) type, whereas most other analysis requires independent variables to be continuous [8].
- MLR does not require inclusion of error terms that are normally distributed [8].

Moreover, MLR allows modeling the likelihood of changing to a particular outcome from the existing outcome if a particular independent variable changes. Therefore, with MLR, we can set a reference CR comment category and analyze the likelihood of another CR comment category if particular independent changes. Following subsections detail our model construction and analysis approach.

6.2.1 Model training and evaluation

For our MLR model, we set *comment_group* as the dependent variable, which denotes a CR comment's group listed in Table 1. As our categorization scheme listed in Table 1 divides CR comments into five categories, this categorical variable has five possible values. We set the 'Functional' category as the reference since comments belonging to this category are a top priority. We excluded the 106 comments belonging to 'Others' for this analysis. We use the `multinom` function of the `nnet` R package to construct our MLR model. We estimate our models' goodness of fit using Nagelkerke *Pseudo* R^2 [55], which is 0.221 for our model. Our Log Likelihood test's results (χ^2 =567.67, $p < 0.001$) suggest that our model is significantly better than a Null model. Our model allocates a total of 60 degrees of freedom for the final model, which is significantly lower than the maximum $\frac{2500}{15} = 166$ recommended degrees of freedom by Harrell Jr. [31]. Therefore, our model is not overfitted.

6.2.2 Model analysis

Table 8 shows the results of our MLR models with *Functional* comment group as the reference. The *OR* column under a particular category indicates the odds of that category occurring instead of the 'Functional' if a variable listed under the first column changes by one unit. $OR > 1$ indicates a higher probability of the target class than the reference (i.e., 'Functional') if the associated attribute increases and vice versa. For example, the *OR* value for the variable log_MR (i.e., log of

Table 8: Identify the factors that distinguish *Functional* class from other classes. OR in blue background indicates a significant higher likelihood of *Functional* comments if that variable increases by one unit and red indicates a significant lower likelihood for *Functional* ones.

Attribute	Discussion		Documentation		False Positive		Refactoring	
	OR	p	**OR**	p	**OR**	p	**OR**	p
log_MR	1.11	**0.028***	1.09	**0.047***	1.13	**0.009****	0.96	0.289
RPT	1.00	0.561	1.00	0.475	1.01	0.296	0.99	0.310
log_PFRC	1.09	0.281	1.05	0.545	0.95	0.471	1.03	0.645
RCS	0.45	**0.007****	0.94	0.537	0.82	0.743	0.96	0.628
log_RCE	0.86	0.068	1.08	0.332	0.84	**0.022***	1.02	0.772
PN	1.00	0.831	1.00	0.704	1.00	0.945	1.02	**0.042***
CV	1.70	0.079	10.25	0.00***	4.21	**0.00*****	0.89	0.705
FUR	0.98	0.016*	0.97	**0.000*****	0.98	**0.027***	0.97	**0.001****
log_RI	1.01	0.677	0.94	0.057	1.06	0.084	1.00	0.887
log_CCR	1.27	**0.002****	1.29	**0.000*****	1.08	0.317	1.23	**0.005****
IBF	1.04	0.846	0.97	0.878	0.97	0.871	1.14	0.483
INF	1.20	0.645	1.31	0.455	0.93	0.844	0.90	0.795
CCY	1.00	0.05	1.00	0.0618	1.00	0.137	1.00	0.722
PDR	1.00	0.483	0.99	**0.026***	0.99	0.215	1.00	0.552
PDL	1.00	0.855	1.00	0.137	1.00	0.171	1.00	0.537

*** , **, and * represent statistical significance at $p < 0.001$, $p < 0.01$, and $p < 0.05$ respectively.

mutual code reviews) under 'Discussion' is 1.11 with a p value of 0.0285 (i.e., significant at $p < 0.05$). Therefore, if log_MR increases by one unit, the likelihood of a 'Discussion' instead of 'Functional' increases by 1.11. Similarly, log_MR is also associated with a higher likelihood of both 'Documentation' and 'False positive.' These results shed further light on our findings from RQ2.A, which found a negative association between log_MR and CR usefulness score.

On the other hand, under the 'Discussion,' OR =0.98 for the variable FUR (file under review) indicates that if the number of files under review increases, the likelihood of comments identifying 'Functional' defects also increases. Similar associations are seen between FUR and the other three categories as well. These findings may not be surprising since prior studies have found changes involving a higher number of files are more likely to include defects [12]. Hence, such changes are more likely to receive 'Functional' comments.

A significant odds reduction for 'Discussion' with RCS (reviewer's codeshare) suggests that the likelihood of discussion over 'Functional' decreases if the reviewer's ownership of the file increases. Our results suggest that log_CCR (i.e., log of code churn) decreases the likelihood of 'Functional' over all four categories. These results support prior findings that reviewers opt for shallow reviews for larger code changes [3]. We notice a negative association between log_RCE (i.e., log of a reviewer's commit experience) and 'False positive,' which suggests that the likelihood of a 'False positive' decreases with increased commit count by the reviewer.

A higher likelihood of 'Refactoring' comments with increased 'PN' (patchset number) suggests that if the number of iterations in a CR increase, the later patches are more likely to receive 'Refactoring' suggestions than 'Functional' ones. We notice the highest change in OR value (i.e., 10.25) for CV (comment volume) under 'Documentation.' This result suggests that if comment volume in a file increases, the odds of suggestions to improve those documentation increases drastically. Moreover, a CV increment increases the likelihood of false positives as well. A lower effort requirement during reviews for suggesting documentation improvements than for 'Functional' ones may be a possible cause.

Surprisingly, we notice PDR (the patch description's readability) improves the odds of 'Functional' over 'Documentation.' These results suggest that well-described patches are more likely to receive 'Functional' than 'Documentation' comments. Finally, although we noticed log_RI (i.e., log of review interval) negatively associating with CR usefulness score in RQ2.A, we did not find any significant association here. These results indicate that delayed reviews do not increase the odds of any particular comment category. Those are more likely due to overloaded reviewers providing delayed yet shallow reviews.

Key takeaways: *The number of mutual reviews between the author and a reviewer, the total number of lines added /modified in a change, and the ratio of lines that are comments are negatively associated with receiving comments identifying 'Functional' defects. The odds of functional defects and their identification increase with the number of files under review. Experienced committers of the projects are less likely to author invalid suggestions. Delayed reviews do not increase the odds of any particular category of comment.*

7 Discussion and Implications

1. Comparison with Bosu et al. [13]: Similar to Bosu *et* al. [13], we found 'Functional' defect identification as the top priority for the CR participants. As the study of Bosu et al. was conducted in a commercial setup (Microsoft), by contrasting the findings of the two studies, we can get a picture of the perspective differences for commercial and OSS projects. Our results deviate from the study of Bosu et al. among several key aspects as described in the following.

- Participants from Bosu *et* al. did not consider questions to understand the implementation as 'Useful'. However, most of our respondents consider questions as useful, with its average 'sample user rating' ranking fourth and 'Avg. Rating' ranking sixth.
- Similarly, participants from Bosu *et* al.'s study consider 'Praise' and 'Design discussion' as 'Not Useful'. However, the majority of our respondents consider 'Design Discussion' as useful. We found 'Praise' as one of the categories with the highest standard deviation, which indicates developers' opinions vary widely regarding this category.
- Bosu *et* al. found that a reviewer's prior experience with the file under review, either as a reviewer or as an author, positively associates with the likelihood of providing useful reviews. However, we found none of the factors to measure these experiences (i.e., RCS, log_PFRC, and log_COMC) having significant associations with usefulness scores in our study.
- Bosu *et* al. found CR usefulness drops as the number of files under review increases. However, we did not notice any such association. On the contrary, we found the likelihood of 'Functional' defects increasing with the number of files.
- Bosu *et* al. found a positive association between the likelihood of useful reviews and reviewers' Microsoft tenure. However, we found a negative association between project tenure and useful reviews. We found the presence of casual contributors [60], which does not apply to Microsoft, as the reason behind this difference.

2. Comparison with Kononeko *et* al. [44, 45]

Kononenko *et* al. conducted a study on Mozilla to understand whether people and participation have an influence on review quality as measured by missed defects [45]. Results of their study suggest that the number of files in a patchset is positively associated with post-review defects, while the reviewer's experience with the project has the opposite association. While we have a different measure, similar to Kononeko *et* al., we found the number of files negatively associated with review usefulness. However, we noticed no significant association between the experience measures and review usefulness. In a subsequent study, Kononenko *et* al. conducted a survey of 88 Mozilla core developers to understand their perspective on the code review quality, factors influencing their code evaluations, and challenges encountered during reviews [44]. One of their research questions, where they investigated the characteristics of a well-done code review, resembles our research question RQ1.A. Although several perspectives of the Mozilla core developers, such as finding defects, being constructive, being clear and thorough, reviewing on time, and sharing knowledge, are similar to ones expressed by the OpenDev developers, we also found several new aspects of review usefulness, which include being respectful, appreciating good work, facilitating better designs, and improving project maintainability.

3. Comparison with other studies investigating CR usefulness: Hasan *et* el. partially replicated Bosu *et* al.'s study at another industrial context (i.e., SRBD). The results of their study concur with ours that 'Functional' defects are top priorities and developers have mixed opinions regarding the usefulness of 'Praise', 'Questions', and 'Documentation'. Rahman *et* al. investigated the relationship between developer experience and CR usefulness. Our results concur with their findings that code-commit experience has positive associations. However, contrasting their findings we do not find any significant association between CR usefulness and authorship/reviewership for the file under review.

3. Reviewer selection: The results of our study suggest that CR usefulness decreases if the number of mutual reviews increases. Our results further suggest that such pairs are likelier to engage in discussions, identify false issues, or suggest trivial documentation-related changes. With a higher number of mutual reviews, two persons are more likely to be aware of each others' strengths and weaknesses and therefore overlook the author's areas of strength. Moreover, such reviews become more predictable and do not bring any fresh perspectives. Although an author is more likely to receive acceptance votes from such peers [72], the results of this study discourage 'Quid pro-Quo' reviews and recommend rotating reviewers, if possible. Moreover, such rotations will also help knowledge decentralization.

The results of our study also suggest that the likelihood of useful reviews increases with the reviewer's commit count. Therefore, experienced authors of a project are the best reviewers, who not only provide useful feedback but also are significantly less likely to write invalid ones. However, this approach has drawbacks due to knowledge centralization as well as overloading experienced contributors. Therefore, we recommend adding such contributors as reviewers for critical code changes.

We also found reviewers who have long tenure with the project but have not contributed many commits (i.e., 'Casual contributors' [60]) being associated with low CR usefulness. While authors may invite such contributors as reviewers for various reasons, they should also include qualified reviewers in such cases to ensure adequate scrutiny.

4. Recommendation to reviewers: CR comments belonging to the 'Functional' group are most useful to authors. Therefore, a reviewer should focus the most on providing such comments. Comments belonging to the 'Refactoring' group rank second. However, 'Documentation', nit-picking, and style issues that can be identified using static analysis tools should get the lowest priority. Finally, instead of keeping an author waiting and providing a shallow review, a reviewer should avoid being in such a scenario by rejecting review invitations if they cannot review on time or spend adequate effort.

5. Recommendation to authors: Our results also suggest that the readability of patch description improves the likelihood of 'Functional' comments than 'Documentation' ones. Therefore, an author should create a meaningful and readable description of the patchset during review preparation. On the contrary, CR usefulness decreases if comment volume increases. Although most authors do not find suggestions to improve 'Documentation' useful, they are more likely to receive those if they write more comments. 'Documentation' comments are low-hanging fruits, and reviewers are more likely to produce those if they do not have time for thorough reviews. Therefore, if an author finds 'Documentation' comments useless, we recommend following the Agile Manifesto guideline [24] by writing self-documenting code and writing documentation only when necessary.

Similar to prior studies [3], we found changes with higher code churns are less likely to receive CR comments identifying 'Functional' changes. Therefore, we recommend committing smaller incremental changes.

Finally, delayed reviews not only frustrate authors but also such reviews are less likely to be helpful. Our results indicate that delayed reviews are less likely to be the results of thorough reviews but more likely due to overloaded reviewers or other factors. Therefore, an author should invite an alternative instead of waiting for a reviewer who is already late beyond reasonable expectation.

6. Using appropriate language: CRs are most useful to developers when participants discuss code issues constructively and empathetically. Constructive conversation helps to maintain a healthy relationship among the teammates [68]. Approximately one-third of respondents indicated appropriateness of the language as an important factor in judging CR usefulness. Inappropriate language may shift the attention from the actual issue (i.e., code under review) to the participant's personal characteristics and, therefore, not only degrade CR's usefulness but can instigate conflicts. Prior studies also have found that inappropriate languages hinder diversity, equity, and inclusion initiatives by disproportionately hurting the participation of women and other minorities [29].

7. Recommendation to researchers: The deviations between our results and the ones reported in prior studies [13,63] suggest that more replications are crucial to understanding how various project or organizational factors influence CR usefulness. Although authorship / reviewership experience for the artifacts under review is the most favored attribute for reviewer recommendation systems [62,76,82], our results indicate no significant association with those with CR usefulness scores. Both Bosu *et* al. Rahman *et* al. found a non-linear relationship between CR usefulness and those experience measures, where CR usefulness increases linearly with experience measures at lower values but plateaus beyond a certain threshold, such as 5-15. As we log-transformed those experience measures (i.e., PFRC and COMC), we expected linear relationships between usefulness score and (log_COMC, log_PFRC). However, we found no significant association between file experience measures and CR usefulness. Although log_COMC was dropped from the model due to multicollinearity with RCS, we did not notice any significant association with RCS either. Hence, if we had trained models with log_COMC instead of RCS, we would have seen a similar association as the one observed for RCS. This result raises whether these attributes are good candidates for building reviewer recommendation systems. While it is premature to make a definite recommendation based on a single study, we recommend more investigations to identify context-specific appropriate attributes for reviewer recommendation systems.

We also notice that attributes that increase the likelihood of the 'Functional' group differ from those increasing 'Discussion' or 'Documentation.' While existing multi-objective reviewer recommendation systems [2, 52] consider expertise, workload, and knowledge distributions, identifying different categories of issues has not been explored. We recommend considering this aspect as another objective for multi-objective reviewer recommendation systems.

8 Related Work

Besides finding defects, known benefits of CR include: improving project maintainability, maintaining code integrity, improving relationships between the participants [11], preventing security defects [12,59] and spreading knowledge, expertise, and development techniques among the review participants [3,64]. Hence, CR has achieved widespread adoption among both commercial OSS development organizations [3,64,67]. Many projects mandate each code change be approved through CR before it can be considered for integration into the project's main code base [64].

Despite significantly associated efforts, most of the CRs do not find bugs [17], as the majority of the CR comments involve maintainability issues that do not impact code output [3]. Defect identification during CRs requires time commitment as well as a deeper understanding of the code context. Reviewers inadequately fulfilling these two criteria often focus more on refactoring or nit-picking issues than defect identification [3]. Several studies have investigated the types of issues identified during CRs [3, 9, 13, 32, 49] and found similar ratios across various commercial as well as OSS projects, with functional defects representing less than 20% of the CR comments.

On a goal to improve CR effectiveness, prior studies have focused on understanding what makes a CR useful to the participants [13, 32, 63]. In this direction, Bosu *et al.* [13] conducted a three-stage empirical study in Microsoft. In the first stage, they interviewed developers to understand what makes a CR useful. Based on the insights obtained from the interviews, they manually labeled a dataset CRs to train an automated model to predict useful reviews. In the third stage, they used the automated model to predict the usefulness of 1.5 million CR comments and conduct an empirical study to understand how CR usefulness varies with various factors [13]. The results of their empirical study suggest that the reviewer's prior experience in changing or reviewing the artifact and the reviewer's project experience increases the likelihood that s/he will provide useful feedback. Hasan *et* al. replicated the first two stages of Bosu *et* al. study in another commercial organization and found that while developers agree on several CR usefulness criteria such as defect identification or solution approach, the same cannot be said for suggestions to improve documentation, praise, or questions asking clarification [32]. While these two studies only focused on the notion of usefulness post-review completion, Rahman *et* al. developed a classifier to predict useful CR comments using both textual and contextual features [63].

Kononenko et al. [45] examined CR quality and found that personal factors such as reviewer workload, experience, and participation factors such as the number of developers involved have significant associations. A later empirical study at Mozilla, by the same set of authors, found that useful CRs are thorough and are influenced not only by the reviewer's familiarity with the code but also by the perceived quality of the code itself [44]. Bosu *et* al.'s survey of OSS and Microsoft developers suggest that several human factors, such as the author's reputation and the relationship between an author and a reviewer, also dictate CR effectiveness [11]. The results of Hatton *et* al.' suggest that the capability to identify defects during CRs may vary widely among reviewers as the best reviewer can be up to 10 times more efficient than the worst ones [34]. Other factors decreasing CR quality include missing rationale, discussion of non-functional requirements of the solution, and lack of familiarity with existing code [19], co-working frequency of a reviewer with the patch author [72], description length of a patch [74], and the level of agreement among the reviewers [38].

Several recent studies have focused on improving CR effectiveness through the automation of various CR tasks. Reviewer recommendation systems [57,62,66,76, 82] focus on finding the best reviewer(s) for a given change. Since understanding changes during CRs are often time-consuming, researchers have proposed tools and frameworks to support changeset comprehension [6, 18,27,41,71]. Changeset size reduction is another automation direction, which aims to decide which change fragments are error-prone and need to be checked in detail to expedite a CR process [7]. Recent studies have focused on automating the reviews through ML-based models [40,75,77,78].

9 Limitations

Internal validity: Our sample selection is the primary threat to internal validity. While we surveyed developers from the entire OpenDev community, we selected CR comments from only the OpenDev Nova project. While Nova is the largest and most active project in this community, we cannot claim that it is an accurate representation of the entire community. However, we are unaware of any evidence regarding the contrary.

Construct Validity: Our survey design and code snippet selection may be subject to biases. To mitigate the bias, we send the survey to two experts in the Software Engineering field and modify the survey based on their feedback. We also told three student researchers to identify any ambiguous questions before sending the survey to the developers.

For our quantitative analyses (i.e., RQ2.A, RQ2.A, RQ2.B), we focused only on comment categories to measure CR usefulness since some of the secondary criteria, such as knowledge sharing and relationship formation, are difficult to evaluate by an independent rater within the limited context of a survey. While we are aware of prior CR studies [15, 52] introducing a metric to measure knowledge sharing, we could not use those metrics since our unit of analyses is individual CR comment, as opposed to the entire CR used in those studies.

Multicollinearity is a potential threat to regression models. Multicollinearity-related threats arise when two independent variables are highly correlated, and due to the interaction between two multicollinear variables, the interaction between an independent and the dependent variables is underestimated. To mitigate this threat, we followed Harrell Jr. [31]'s approach to identifying highly correlated variable clusters and picked only one variable from a cluster to train our models.

External Validity: We divided our study objectives into two research questions. The first question aims to understand OSS developers' perspectives about the usefulness of CRs and comment categories that are considered most useful to the developers. Although we conducted our survey on OpenDev developers, most of the developers have experience working on other open-source and industrial projects. Therefore our findings might apply to other OSS and industrial projects. However, as OSS projects vary based on technology, norms, culture, and governance, external validity remains a threat.

Our second research question aims to identify the factors that are associated with CR usefulness. Since this analysis includes only one project's data, this result may not be generalized outside of our study subject. Manually labeling a large dataset using multiple raters is time-consuming, especially when we have 18 different choices for each instance. Therefore, we were unable to include multiple projects' datasets in this analysis. Replications of this study are essential to derive context-specific customized CR-specific recommendations. To promote replications, we have made our survey questions, anonymized dataset, and analysis scripts publicly available.

Conclusion validity: While constructing a regression model, overfitting appears to be the primary threat to conclusion validity. We followed Harrell Jr.'s recommendations to encounter this threat by allocating degrees of freedom less than (n/15), where n is the size of the dataset. For model training, we use `stats` and `nnet` packages, which are considered the gold standards for regression modeling. Hence, we do not anticipate any threat to library selections. Finally, we noticed a low R^2 value for our linear regression model due to the nature of the code review dataset. However, as our goal is to train inferential models, this low measure does not invalidate obtained insights. Moreover, our log-likelihood tests found this model significantly better than a null model.

10 conclusion

We conducted a three-stage mixed-method study investigating CR's usefulness among OSS developers. We manually categorized 2,500 CR comments, using those comments, designed an online survey, received 160 usable responses from OpenDev developers, combined insights obtained from the survey with our manually labeled dataset, and finally trained two regression models to provide a better understanding of what makes code review useful and the set of factors influencing CR usefulness. The results of our study suggest that a CR comment's usefulness is dictated not only by its technical contributions, such as defect findings or quality improvement tips but also by its linguistic characteristics, such as comprehensibility and politeness. While a reviewer's coding experience positively associates with CR usefulness, the number of mutual reviews, comment volume in a file, the total number of lines added /modified, and CR interval have the opposite associations. While authorship and reviewership experiences for the files under review have been the most popular attributes for reviewer recommendation systems, we do not find any significant association of those attributes with CR usefulness. As we find several of our results deviating from prior studies, we also recommend more investigations to identify context-specific attributes for reviewer recommendation models.

Data availability

Our analysis scripts and aggregated dataset are publicly available at: https://github.com/WSU-SEAL/CR-usefulness-EMSE. Upon acceptance, we plan to post it on Zenodo with a permanent DOI.

Funding and Conflicts of interests/Competing interests.

Work conducted for this research is partially supported by the US National Science Foundation under Grant No. 1850475. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

We thank Hemangi Murdande for her assistance during manual data labeling.

The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Allison, P.: Prediction vs. causation in regression analysis. Statistical Horizons **703** (2014)
2. Asthana, S., Kumar, R., Bhagwan, R., Bird, C., Bansal, C., Maddila, C., Mehta, S., Ashok, B.: Whodo: automating reviewer suggestions at scale. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 937–945 (2019)
3. Bacchelli, A., Bird, C.: Expectations, outcomes, and challenges of modern code review. In: 2013 35th International Conference on Software Engineering (ICSE), pp. 712–721. IEEE (2013)
4. Balachandran, V.: Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In: 2013 35th International Conference on Software Engineering (ICSE), pp. 931–940. IEEE (2013)
5. Baltes, S., Diehl, S.: Worse than spam: Issues in sampling software developers. In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 1–6 (2016)
6. Barnett, M., Bird, C., Brunet, J., Lahiri, S.K.: Helping developers help themselves: Automatic decomposition of code review changesets. In: Proceedings of the 37th International Conference on Software Engineering-Volume 1, pp. 134–144. IEEE Press (2015)
7. Baum, T., Schneider, K.: On the need for a new generation of code review tools. In: International Conference on Product-Focused Software Process Improvement, pp. 301–308. Springer (2016)
8. Bayaga, A.: Multinomial logistic regression: Usage and application in risk analysis. Journal of applied quantitative methods **5**(2) (2010)
9. Beller, M., Bacchelli, A., Zaidman, A., Juergens, E.: Modern code reviews in open-source projects: Which problems do they fix? In: Proceedings of the 11th working conference on mining software repositories, pp. 202–211 (2014)
10. Benjamini, Y., Hochberg, Y.: Controlling the false discovery rate: a practical and powerful approach to multiple testing. Journal of the Royal statistical society: series B (Methodological) **57**(1), 289–300 (1995)
11. Bosu, A., Carver, J.C., Bird, C., Orbeck, J., Chockley, C.: 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 (2017)
12. Bosu, A., Carver, J.C., Hafiz, M., Hilley, P., Janni, D.: Identifying the characteristics of vulnerable code changes: An empirical study. In: 22nd ACM SIGSOFT International Symposium on the Foundations of Software Engineering, FSE '14, pp. 257–268. Hong Kong, China (2014)
13. Bosu, A., Greiler, M., Bird, C.: Characteristics of useful code reviews: An empirical study at microsoft. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, pp. 146–156. IEEE (2015)
14. Changyong, F., Hongyue, W., Naiji, L., Tian, C., Hua, H., Ying, L., et al.: Log-transformation and its implications for data analysis. Shanghai archives of psychiatry **26**(2), 105 (2014)
15. Chouchen, M., Ouni, A., Mkaouer, M.W., Kula, R.G., Inoue, K.: Whoreview: A multi-objective search-based approach for code reviewers recommendation in modern code review. Applied Soft Computing **100**, 106908 (2021)
16. Cohen, J.: A coefficient of agreement for nominal scales. Educational and psychological measurement **20**(1), 37–46 (1960)
17. Czerwonka, J., Greiler, M., Tilford, J.: Code reviews do not find bugs. how the current code review best practice slows us down. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol. 2, pp. 27–28. IEEE (2015)
18. Dias, M., Bacchelli, A., Gousios, G., Cassou, D., Ducasse, S.: Untangling fine-grained code changes. In: 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), pp. 341–350. IEEE (2015)
19. Ebert, F., Castor, F., Novielli, N., Serebrenik, A.: An exploratory study on confusion in code reviews. Empirical Software Engineering **26**, 1–48 (2021)
20. Fagan, M.E.: Design and code inspections to reduce errors in program development. IBM Syst. J. **15**(3), 182–211 (1976). DOI 10.1147/sj.153.0182. URL https://doi.org/10.1147/sj.153.0182
21. Foley, B.: What is regression analysis and why should i use it. Source: https://www. surveygizmo. com/resources/blog/regression-analysis (2018)
22. Foundation, O.: 2022 openifra foundation annual report. https://openinfra.dev/annual-report/2022 (2022)
23. Fowler, M.: Refactoring catalog. Refactoring Home Page, URL: http://www. refactoring. com/catalog/index. html (letzter Abruf: 09.02. 2006) (2012)
24. Fowler, M., Highsmith, J., et al.: The agile manifesto. Software development **9**(8), 28–35 (2001)
25. Fukushima, T., Kamei, Y., McIntosh, S., Yamashita, K., Ubayashi, N.: An empirical study of just-in-time defect prediction using cross-project models. In: Proceedings of the 11th Working Conference on Mining Software Repositories, pp. 172–181 (2014)

26. Gauthier, I.X., Lamothe, M., Mussbacher, G., McIntosh, S.: Is historical data an appropriate benchmark for reviewer recommendation systems?: A case study of the gerrit community. In: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 30–41. IEEE (2021)
27. Gómez, V.U., Ducasse, S., D'Hondt, T.: Visually characterizing source code changes. Science of Computer Programming **98**, 376–393 (2015)
28. Governance, O.: Openstack project teams. https://governance.openstack.org/tc/reference/projects/ (2023)
29. Gunawardena, S.D., Devine, P., Beaumont, I., Garden, L.P., Murphy-Hill, E., Blincoe, K.: Destructive criticism in software code review impacts inclusion. Proceedings of the ACM on Human-Computer Interaction **6**(CSCW2), 1–29 (2022)
30. Han, X., Tahir, A., Liang, P., Counsell, S., Luo, Y.: Understanding code smell detection via code review: A study of the openstack community. In: 2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC), pp. 323–334. IEEE (2021)
31. Harrell Jr, F.E., Lee, K.L., Califf, R.M., Pryor, D.B., Rosati, R.A.: Regression modelling strategies for improved prognostic prediction. Statistics in medicine **3**(2), 143–152 (1984)
32. Hasan, M., Iqbal, A., Islam, M.R.U., Rahman, A., Bosu, A.: Using a balanced scorecard to identify opportunities to improve code review effectiveness: an industrial experience report. Empirical Software Engineering **26**(6), 1–34 (2021)
33. Hassan, A.E., Holt, R.C.: Studying the chaos of code development. In: WCRE, vol. 3, p. 123 (2003)
34. Hatton, L.: Testing the value of checklists in code inspections. IEEE software **25**(4), 82–88 (2008)
35. Helland, I.S.: On the interpretation and use of r2 in regression analysis. Biometrics pp. 61–69 (1987)
36. Henley, A.Z., Muçlu, K., Christakis, M., Fleming, S.D., Bird, C.: Cfar: A tool to increase communication, productivity, and review quality in collaborative code reviews. In: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, pp. 1–13 (2018)
37. Hinkle, D., Jurs, H., Wiersma, W.: Applied statistics for the behavioral sciences (1998)
38. Hirao, T., Ihara, A., Ueda, Y., Phannachitta, P., Matsumoto, K.i.: The impact of a low level of agreement among reviewers in a code review process. In: IFIP International Conference on Open Source Systems, pp. 97–110. Springer (2016)
39. Hirao, T., McIntosh, S., Ihara, A., Matsumoto, K.: Code reviews with divergent review scores: An empirical study of the openstack and qt communities. IEEE Transactions on Software Engineering (2020)
40. Hong, Y., Tantithamthavorn, C., Thongtanunam, P., Aleti, A.: Commentfinder: a simpler, faster, more accurate code review comments recommendation. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 507–519 (2022)
41. Huang, Y., Jia, N., Chen, X., Hong, K., Zheng, Z.: Code review knowledge perception: Fusing multi-features for salient-class location. IEEE Transactions on Software Engineering pp. 1–1 (2020). DOI 10.1109/TSE.2020.3021902
42. Jiarpakdee, J., Tantithamthavorn, C., Treude, C.: Autospearman: Automatically mitigating correlated software metrics for interpreting defect models. In: 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 92–103. IEEE Computer Society (2018)
43. Kincaid, J.P., Fishburne Jr, R.P., Rogers, R.L., Chissom, B.S.: Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel. Tech. rep., Naval Technical Training Command Millington TN Research Branch (1975)
44. Kononenko, O., Baysal, O., Godfrey, M.W.: Code review quality: How developers see it. In: Proceedings of the 38th International Conference on Software Engineering, ICSE '16, pp. 1028–1038. ACM, New York, NY, USA (2016). DOI 10.1145/2884781.2884840. URL http://doi.acm.org/10.1145/2884781.2884840
45. Kononenko, O., Baysal, O., Guerrouj, L., Cao, Y., Godfrey, M.W.: Investigating code review quality: Do people and participation matter? In: 2015 IEEE international conference on software maintenance and evolution (ICSME), pp. 111–120. IEEE (2015)
46. Kononenko, O., Rose, T., Baysal, O., Godfrey, M., Theisen, D., De Water, B.: Studying pull request merges: a case study of shopify's active merchant. In: Proceedings of the 40th international conference on software engineering: software engineering in practice, pp. 124–133 (2018)
47. Landis, J.R., Koch, G.G.: An application of hierarchical kappa-type statistics in the assessment of majority agreement among multiple observers. Biometrics pp. 363–374 (1977)
48. Mansfield, E.R., Helms, B.P.: Detecting multicollinearity. The American Statistician **36**(3a), 158–160 (1982)
49. Mäntylä, M.V., Lassenius, C.: What types of defects are really discovered in code reviews? IEEE Transactions on Software Engineering **35**(3), 430–448 (2008)
50. McCabe, T.J.: A complexity measure. IEEE Transactions on software Engineering (4), 308–320 (1976)
51. McIntosh, S., Kamei, Y., Adams, B., Hassan, A.E.: An empirical study of the impact of modern code review practices on software quality. Empirical Software Engineering **21**, 2146–2189 (2016)
52. Mirsaeedi, E., Rigby, P.C.: Mitigating turnover with code review recommendation: balancing expertise, workload, and knowledge distribution. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, pp. 1183–1195 (2020)
53. Nagappan, N., Ball, T.: Use of relative code churn measures to predict system defect density. In: Proceedings of the 27th international conference on Software engineering, pp. 284–292 (2005)
54. Nagappan, N., Ball, T.: Using software dependencies and churn metrics to predict field failures: An empirical case study. In: First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007), pp. 364–373. IEEE (2007)
55. Nagelkerke, N.J., et al.: A note on a general definition of the coefficient of determination. Biometrika **78**(3), 691–692 (1991)
56. Osborne, J.W., Waters, E.: Four assumptions of multiple regression that researchers should always test. Practical assessment, research, and evaluation **8**(1), 2 (2002)
57. Pandya, P., Tiwari, S.: Corms: A github and gerrit based hybrid code reviewer recommendation approach for modern code review. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, p. 546–557. Association for Computing Machinery, New York, NY, USA (2022)

58. Panichella, S., Zaugg, N.: An empirical investigation of relevant changes and automation needs in modern code review. Empirical Software Engineering **25**(TBD), TBD (2020)
59. Paul, R., Turzo, A.K., Bosu, A.: Why security defects go unnoticed during code reviews? a case-control study of the chromium os project. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp. 1373–1385. IEEE (2021)
60. Pinto, G., Steinmacher, I., Gerosa, M.A.: More common than you think: An in-depth study of casual contributors. In: 2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER), vol. 1, pp. 112–123. IEEE (2016)
61. Qualtrics: Question types. https://www.qualtrics.com/support/survey-platform/survey-module/editing-questions/question-types-guide/question-types-overview/
62. Rahman, M.M., Roy, C.K., Collins, J.A.: Correct: code reviewer recommendation in github based on cross-project and technology experience. In: Proceedings of the 38th International Conference on Software Engineering Companion, pp. 222–231 (2016)
63. Rahman, M.M., Roy, C.K., Kula, R.G.: Predicting usefulness of code review comments using textual features and developer experience. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), pp. 215–226. IEEE (2017)
64. Rigby, P.C., Bird, C.: Convergent contemporary software peer review practices. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, pp. 202–212 (2013)
65. Rigby, P.C., Storey, M.A.: Understanding broadcast based peer review on open source software projects. In: 2011 33rd International Conference on Software Engineering (ICSE), pp. 541–550. IEEE (2011)
66. Rong, G., Zhang, Y., Yang, L., Zhang, F., Kuang, H., Zhang, H.: Modeling review history for reviewer recommendation: A hypergraph approach. In: Proceedings of the 44th International Conference on Software Engineering, ICSE '22, p. 1381–1392. Association for Computing Machinery, New York, NY, USA (2022)
67. Sadowski, C., Söderberg, E., Church, L., Sipko, M., Bacchelli, A.: Modern code review: a case study at google. In: Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice, pp. 181–190 (2018)
68. Sarker, J., Turzo, A.K., Bosu, A.: A benchmark study of the contemporary toxicity detectors on software engineering interactions. arXiv preprint arXiv:2009.09331 (2020)
69. Sarle, W.: Sas/stat user's guide: The varclus procedure. sas institute. Inc., Cary, NC, USA (1990)
70. Snow, J., Mann, M.: Qualtrics survey software: handbook for research professionals. Qualtrics Labs, Inc (2013)
71. Tao, Y., Kim, S.: Partitioning composite code changes to facilitate code review. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, pp. 180–190. IEEE (2015)
72. Thongtanunam, P., Hassan, A.E.: Review dynamics and their impact on software quality. IEEE Transactions on Software Engineering **47**(12), 2698–2712 (2020)
73. Thongtanunam, P., McIntosh, S., Hassan, A.E., Iida, H.: Investigating code review practices in defective files: An empirical study of the qt system. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, pp. 168–179. IEEE (2015)
74. Thongtanunam, P., McIntosh, S., Hassan, A.E., Iida, H.: Review participation in modern code review. Empirical Software Engineering **22**(2), 768–817 (2017)
75. Thongtanunam, P., Pornprasit, C., Tantithamthavorn, C.: Autotransform: automated code transformation to support modern code review process. In: Proceedings of the 44th International Conference on Software Engineering, pp. 237–248 (2022)
76. Thongtanunam, P., Tantithamthavorn, C., Kula, R.G., Yoshida, N., Iida, H., Matsumoto, K.i.: Who should review my code? a file location-based code-reviewer recommendation approach for modern code review. In: 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), pp. 141–150. IEEE (2015)
77. Tufan, R., Pascarella, L., Tufanoy, M., Poshyvanykz, D., Bavota, G.: Towards automating code review activities. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp. 163–174. IEEE (2021)
78. Tufano, R., Masiero, S., Mastropaolo, A., Pascarella, L., Poshyvanyk, D., Bavota, G.: Using pre-trained models to boost code review automation. In: Proceedings of the 44th International Conference on Software Engineering, pp. 2291–2302 (2022)
79. Yamane, T.: Statistics: an introductory analysis-3 (1973)
80. Zaidman, A., Van Rompaey, B., Van Deursen, A., Demeyer, S.: Studying the co-evolution of production and test code in open source and industrial developer test processes through repository mining. Empirical Software Engineering **16**, 325–364 (2011)
81. Zanaty, F.E., Hirao, T., McIntosh, S., Ihara, A., Matsumoto, K.: An empirical study of design discussions in code review. In: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 1–10 (2018)
82. Zanjani, M.B., Kagdi, H., Bird, C.: Automatically recommending peer reviewers in modern code review. IEEE Transactions on Software Engineering **42**(6), 530–543 (2015)