Understanding Practitioners' Expectations on Clear Code Review Comments

JUNKAI CHEN*, Zhejiang University, China
ZHENHAO LI*, York University, Canada
QIHENG MAO, Zhejiang University, China
XING HU†, Zhejiang University, China
KUI LIU, Zhejiang University, China
XIN XIA, Zhejiang University, China

WARNING: This paper contains potentially offensive and harmful content.
The code review comment (CRC) is pivotal in the process of modern code review. It provides reviewers with the opportunity to identify potential bugs, offer constructive feedback, and suggest improvements. Clear and concise code review comments (CRCs) facilitate the communication between developers and are crucial to the correct understanding of the identified issues and proposed solutions. Despite the importance of CRCs' clarity, there is still a lack of guidelines on what constitutes a good clarity and how to evaluate it. In this paper, we conduct a comprehensive study on understanding and evaluating the clarity of CRCs. We first derive a set of attributes related to the clarity of CRCs, namely RIE attributes (i.e., *Relevance*, *Informativeness*, and *Expression*), as well as their corresponding evaluation criteria based on our literature review and survey with practitioners. We then investigate the clarity of CRCs in open-source projects written in nine programming languages and find that a large portion (i.e., 28.8%) of the CRCs lack the clarity in at least one of the attributes. Finally, we explore the potential of automatically evaluating the clarity of CRCs by proposing ClearCRC. Experimental results show that ClearCRC with pre-trained language models is promising for effective evaluation of the clarity of CRCs, achieving a balanced accuracy up to 73.04% and a F-1 score up to 94.61%.

CCS Concepts: • **Software and its engineering → Software creation and management**.

Additional Key Words and Phrases: Clarity, Code Review Comment, Code Review

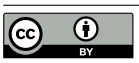
ACM Reference Format:
Junkai Chen, Zhenhao Li, Qiheng Mao, Xing Hu, Kui Liu, and Xin Xia. 2025. Understanding Practitioners' Expectations on Clear Code Review Comments. *Proc. ACM Softw. Eng.* 2, ISSTA, Article ISSTA056 (July 2025), 23 pages. https://doi.org/10.1145/3728931

1 Introduction

Code review is the process of systematic examinations on software source code performed by third-party developers [42, 49]. The primary goals of code review include identifying potential issues, seeking areas for improvement, and transferring knowledge [9, 30, 51, 63, 65]. It has been

* Equal contribution.
† Corresponding author.

Authors' Contact Information: Junkai Chen, The State Key Laboratory of Blockchain and Data Security, Zhejiang University, Hangzhou, China, junkaichen@zju.edu.cn; Zhenhao Li, York University, Toronto, Canada, lzhenhao@yorku.ca; Qiheng Mao, Zhejiang University, Hangzhou, China, maoqiheng@zju.edu.cn; Xing Hu, The State Key Laboratory of Blockchain and Data Security, Zhejiang University, Hangzhou, China, xinghu@zju.edu.cn; Kui Liu, Zhejiang University, Hangzhou, China, brucekuiliu@gmail.com; Xin Xia, Zhejiang University, Hangzhou, China, xin.xia@acm.org.



ACM 2994-970X/2025/7-ARTISSTA056
https://doi.org/10.1145/3728931

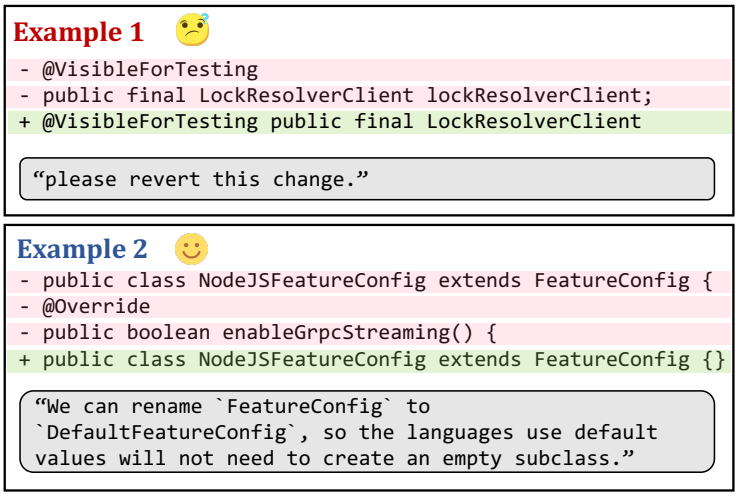


Fig. 1. Examples of code changes and their code review comments (CRCs).

widely integrated into the software development life cycle in both open-source and industrial projects to help the assurance of software quality.

A code review comment (CRC) is a specific piece of feedback provided by a reviewer during the code review process. Clear and concise code review comments (CRCs) are crucial for ensuring that the feedback is readable and actionable, and further contributing to the overall quality of the software. On the contrary, CRCs that lack of sufficient clarity may lead to confusion, misunderstandings, and misinterpretations amongst the collaborating developers. Figure 1 presents two examples of code changes and their corresponding code review comments. In Example 1, the reviewer comments *"please revert this change"*. However, no rationale or reason behind this comment is provided. Developers may not understand why this change needs to be reverted. In Example 2, the reviewer suggests renaming a parent class and also explains the reason of such suggestion. Moreover, this comment is written in a more friendly tone (i.e., *"We can ..."*). Although both of these two examples provide a suggestion to modify the code, their effectiveness in conveying reviewer's idea may considerably vary.

Prior studies provide preliminary insights on revealing the quality of CRCs. For example, usefulness [11, 27, 47] focuses on whether the CRCs can trigger subsequent code changes or if the reply to CRCs has a positive sentiment (e.g., *"LGTM"*). Yang et al. [62] proposed four attributes to evaluate the quality of CRCs. Such attributes focus more on the purpose of CRCs (e.g., evaluation, suggestion, and question). However, these studies either indirectly evaluate the quality of CRCs using the information after the completion of the review, or evaluate the CRCs based on whether it has elements related to the purpose of this comment. Therefore, a systematic understanding and characterizing of how a CRC can clearly and concisely foster the communication among developers (i.e., clarity of CRCs) is still in mystery and an on-going challenge.

In this paper, we conduct a comprehensive study to uncover the clarity of CRCs by following a multi-phased investigation: 1) we understand the characteristics and evaluation criteria of CRCs' clarity through a systematic literature review, a preliminary review with industrial professionals, and an online questionnaire survey with practitioners; 2) we examine the clarity of CRCs in open-source projects by conducting a manual investigation on sampled datasets; 3) we seek to automatically evaluate the clarity of CRCs by proposing an automated framework.

Particularly, we study the clarity of code review comments by answering three research questions:

RQ1: What attributes are relevant to the clarity of CRC? Based on the analysis on our literature review and 103 survey responses from practitioners, we derive our RIE attributes for the clarity of CRCs (i.e., *Relevance*, *Informativeness*, and *Expression*) and their corresponding evaluation criteria. More than 75% of the participants consider that these attributes are important to the clarity of CRCs.

RQ2: How is the clarity of code review comments in open-source projects? We manually investigate the clarity of code review comments in open-source projects using the datasets sampled from the work of Li et al. [33]. We find that 28.8% of the CRCs in our study datasets are insufficient in at least one of the three attributes of CRCs' clarity. Among these attributes, *Informativeness* has the most considerable insufficiency.

RQ3: Can we automatically evaluate the clarity of code review comments? We propose ClearCRC, an automated framework for the evaluation of CRCs' clarity based on the RIE attributes. We compare the results of ClearCRC with three sets of backbone models: 1) training deep learning and machine learning models; 2) fine-tuning pre-trained language models; and 3) prompting large language models (LLMs). Our results show that ClearCRC is effective in evaluating each of the RIE attributes, with an average balanced accuracy of 73.04% using pre-trained language models.

We summarize the contributions of this paper as follows:

- We derive RIE attributes and their corresponding evaluation criteria for the clarity of CRCs by analyzing the results of our literature review and 103 survey responses from practitioners around the world.
- We find that a large portion of the CRCs in open source projects actually lack of sufficient clarity. We also publicly share our manually labelled data in the replication package [1] for future studies.
- We propose ClearCRC, an automated framework for the evaluation of CRCs' clarity using various backbone models such as machine learning, deep learning, pre-trained language models, and large language models. ClearCRC achieves promising results in our evaluation, especially using pre-trained language models.

Overall, the findings of our studies may be used as actionable guidelines for evaluating and writing clear CRCs, as well as for curating high-quality data to improve the automated generation techniques of CRCs.

Paper Organization. Section 2 summarizes the related work. Section 3 presents the methodology of our study. Section 4 discusses the results of our research questions. Section 5 discusses the implications of our study. Section 6 discusses the threats to validity. Section 7 concludes the paper.

2 Related Work

In this section, we summarize the related work in two aspects: studying the quality of code review comments and automated generation of code review comments.

2.1 Quality of Code Review Comments.

Kerzazi et al. [7] found that sentiment conveyed within comments can significantly impact the outcome of the review process. Kononenko et al. [29] suggested that the review quality is mainly associated with the thoroughness of the feedback, the reviewer's familiarity with the code, and the perceived quality of the code itself. Rahman et al. [47] presented a comparative analysis of useful versus non-useful review comments, distinguishing them through their textual attributes and the reviewers' expertise. Comments were classified as useful or non-useful depending on their capacity to instigate changes. Chouchen et al. [16] synthesized negative examples of code reviews

that degraded software quality, categorizing erroneous practices into five patterns: Confused reviewers, Divergent reviewers, Low review participation, Shallow review, and Toxic review. Ram et al. [48] focused on the issue of code change reviewability, which is closely related to the quality of reviews. Ebert et al. [19, 20] identified confusion as a significant detriment to the quality of code reviews and offered recommendations for addressing issues of confusion. Bosu et al. [11] emphasized that the usefulness of code review lies in its ability to assist developers in avoiding defects, adhering to team conventions, and resolving issues efficiently and reasonably. Ferreira et al. [23] highlighted the prevalence of uncivil behavior during the code review process, noting that discourteous comments can hinder project communication and discussion, ultimately slowing down development progress. Pascarella et al. [45] examined the essential information required by reviewers in code reviews, including the suitability of an alternative solution, correct understanding, rationale, code context, etc. Yang et al. [62] introduced four attributes for evaluating the quality of CRC: questions, suggestions, evaluations, and emotion, advocating for an assessment of the quality.

Prior studies provide insights on the quality of CRCs from different perspectives. These studies generally emphasize the importance of *"clear"* CRCs. However, the understanding and characterizing on what are *"clear"* CRCs are still limited. Therefore, in this paper, we conduct a comprehensive study to uncover and demystify what are the characteristics of a *"clear"* CRC.

2.2 Automated Generation of Code Review Comments.

The automated generation of code review comments has been widely studied in recent years [34, 40, 53, 58], aiming to streamline this critical yet often labor-intensive activity in the software engineering life cycle. Efforts in this domain can be categorized into three main types of approaches: traditional rule-based methods, deep learning techniques, and Large Language Models (LLMs) based techniques. Early automation efforts in code review were aimed at identifying code violations and defects utilizing traditional rule-based static analysis tools [8]. While providing a base for automation, they lacked the flexibility to adapt to the nuanced and evolving nature of software development practices. The emergence of deep learning has significantly enhanced the capability to automate code reviews, offering a nuanced understanding and interpretation of code changes. Techniques leveraging LSTM [26], and Transformers [34, 57] have been pivotal in predicting review necessities and generating context-specific feedback. Li et al. [34] proposed a pre-trained model based on the Text-To-Text-Transfer Transformer (T5) model [46], specifically tailored for the code review process across three different code review tasks including the generation of CRCs. LLaMA-Reviewer [40] utilized the LLaMA model and parameter-efficient fine-tuning to automate code review comments generation, achieving performance on par with existing code-review-focused models using fewer resources.

Prior studies commonly incorporated the CRCs data directly, without a curation or quality selection process. Given this scenario, existing CRCs generation techniques might inadvertently learn from CRCs data lacking clarity, resulting in perplexing outcomes. Our study can complement the research of automated CRCs generation to curate the training data, and further improve the quality of the generated CRCs.

3 Methodology

Figure 2 presents an overview of our study. To address the three research questions proposed in the Introduction, we conduct a comprehensive study that involves three phases. ***Phase 1:*** We first derive an initial list of attributes and evaluation criteria concerning the clarity of CRCs through literature review and a preliminary review with industrial professionals. We then survey with practitioners for their perspectives to refine the attributes and evaluation criteria. ***Phase 2:*** We manually investigate the clarity of CRCs in open-source projects using the clarity attributes and

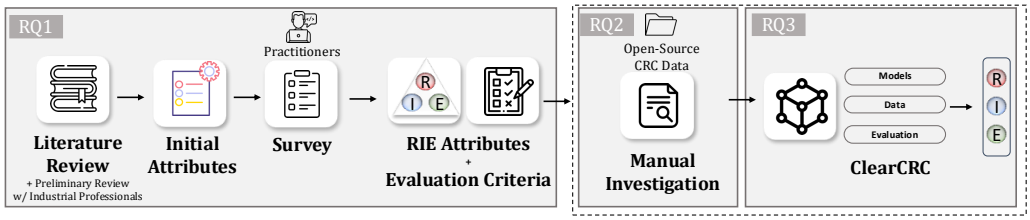


Fig. 2. Overview of our study.

evaluation criteria derived in the prior phase. ***Phase 3:*** We propose ClearCRC, an automated framework that seeks to evaluate the clarity of CRCs based on the attributes derived from our literature review and practitioners' feedback.

3.1 Characterizing the Clarity of CRCs

We characterize the attributes related to the clarity of CRCs by combining 1) a systematic literature review followed by a preliminary review with industrial professionals, and 2) the analysis of our online survey with practitioners.

3.1.1 Literature Review. We analyze existing studies on the quality of CRCs and derive an initial set of attributes related to the clarity of CRCs.

Literature Collection. We first gather the papers related to code review by utilizing the list of papers provided by prior literature reviews [17, 60]. These two literature reviews summarized 139 and 112 prior works on code review published from 2005 to 2019 and 2011 to 2019, respectively. We then follow the strategy of paper collection in these literature reviews to collect papers published after 2019 and collect 70 papers in this process.

Data Analysis. We first read the titles and abstracts of all the collected papers and filter papers that are not related to the studies on CRCs. The topics of such filtered papers include studying the code change, authors, reviewers, pull requests, and commit messages. After this process, we have a list of 47 papers that are related to CRCs for further analysis. Two authors of this paper then independently read these papers and generate an initial set of codes related to the attribute of CRCs' clarity. The authors then perform open card sorting [54] on the generated codes to analyze the codes and sort the generated codes into potential themes that indicate the attributes related to the clarity of CRCs. Particularly, author 1 generates six initial codes: *Confused Reviews, Toxic Reviews, Relevant Reviews, Readable Reviews, Shallow Reviews, and Informative Reviews.* Author 2 generates five initial codes: *Readability, Sentiment, Relevance, Information, and Toxicity.* The two authors then engage in thorough discussions to refine the attribute set. Both authors collaboratively re-evaluate the attributes and reconcile differences by focusing on semantic consistency and conceptual clarity. Throughout this process, attributes that exhibit overlap or ambiguity are merged or redefined. For example, *Readability and Toxicity* are merged into *Expression*, and the meaning of *Shallow Reviews* can be represented by *Informativeness.* Eventually, we derive three attributes that are related to the clarity of CRCs, including ***Relevance***, ***Informativeness***, and ***Expression***. The authors then discuss and generate an initial definition for each attribute.

Preliminary Review. To preliminarily verify the attributes derived from our literature review, we conduct a group interview with 11 industrial practitioners to obtain their feedback. The participants are full-time engineers at Company X, which is a world-leading IT company. All of the participants have at least three years of experience in code review and software development. The duration of this group interview is around 1 hour. We follow a three-step process to conduct the group interview:

1) We ask the participants to freely talk about their expectations on the clarity of CRCs without the knowledge of our derived attributes; 2) We present our derived attributes to the participants; 3) We discuss with the participants for whether the attributes can reflect their expectations on the clarity of CRCs. Eventually, we find that most of the points initially proposed by the participants can be reflected by our derived attributes. The remaining non-reflected points are related to the process of code review rather than the code review comments, including *deciding proper reviewers* and *prompt response time*. At the end, the participants are thanked and briefly informed about the next plans. The participants also provide suggestions regarding the design of surveys. We take their suggestions into consideration while designing our survey in the next step.

3.1.2 Questionnaire Survey with Practitioners. We conduct an online questionnaire survey with practitioners for their perspectives on the clarity of CRCs and further refine the attributes and derive the evaluation criteria.

Survey Design. The questions in our survey are divided into three parts:

- P1 We ask the participants for their basic information (e.g., country or region of residence, primary job role, and years of experience in the primary job role) and if they have experience in code review.
- P2 For each attribute of the clarity, we ask the participants for "*To what extent do you think that the aspect of "{attribute}" is important to the clarity of code review comments?*" where *{attribute}* is replaced with a specific one. The participants then choose from "Very important", "Important", "Neutral", "Unimportant", and "Very unimportant". If "Very important" or "important" is chosen, we further ask the participants for the details of how they will evaluate the corresponding attribute (e.g., what factors and detailed information they may focus on). We will derive the evaluation criteria based on their comments. If other options are chosen, we ask the participant for the potential reasons. At the end of this part, we further ask the participant for "*Do you have some ideas about other important aspects that contribute to the clarity of code review comments?*".
- P3 We ask the participants for questions on automated code review comment generation, such as their experience in using such tools, and their perspectives on the clarity of the generated CRCs.

Survey Implementation. We implement the survey following the design discussed above using Microsoft Forms [4]. We conduct a pilot survey with three practitioners to collect their feedback on the design of our survey. All the practitioners in the pilot survey have experience in writing and reviewing CRCs. The pilot participants provide suggestions regarding the clarification of instructions and the consistency of some terms. We make modifications according to their feedback and have a final version of the survey, which is an anonymous questionnaire and can be accessed using the link provided in our email sent to the participants. A sample of the complete survey is available in our replication package [1].

Participants. To invite participants from diverse backgrounds, we reach out to industrial and academic professionals residing in various countries or regions, across five continents around the world. Eventually, we receive 112 responses in total. We then filter 9 responses of which indicate as having no experience in code review, resulting in 103 remaining responses for further analysis.
➤ *Demographics.* Table 1 shows the statistical information of the participants. The participants reside in 37 countries or regions across five continents, including 49 participants in Europe, 22 participants in Asia, 24 participants in North America, 4 participants in South America, and 4 participants in Oceania. A majority of the participants have an occupation of industrial/freelance professional (76.7%) and primary job role as development (81.6%). A large percentage of the participants (68.9%)

Table 1. Statistics of our survey participants.

Residency		Occupation		Primary Job Role		Experience	
Asia	22	Academic or industrial researcher	8	Development	84	[0-2]	10
Europe	49	Graduate or undergraduate student	12	Software Project Management	5	[3-5]	22
North America	24	Industrial or freelance professional	79	Testing	2	[6-9]	19
Oceania	4	Other	4	Research	9	[10+]	52
South America	4	-	-	Other	3	-	-

have at least five years of experience in their primary job role. In short, our survey participants reside in various countries or regions all over the world, and most of them are industrial or freelance professionals with more than 5 years of experience in software development.

Data Analysis. The data we obtain from the survey consists of option data from multiple-choice questions and natural language response data from open-ended questions. (1) For the questions of multiple choices, we compute the percentage of each option, e.g., the percentage of survey participants who think "Relevance" is "Very Important" to the clarity of CRCs shown in Fig. 3. (2) For the open questions (e.g., details for evaluating the attributes), we generate codes from the answers and perform open card sorting [54] to analyze the thematic similarity. Specifically, to derive the evaluation criteria for each attribute, we first extract and record criteria of all the responses, and then sort and categorize them into concise and explicit descriptions like "Proper syntax and grammar". For example, a response from one survey participant for *Informativeness* said "*Explain why, with specific reference to the change.*" will finally lead to two criteria including "I.E2: Provide reasons or context information" and "I.O2: Provide reference information" (See Section 4). When a consensus on these criteria for each attribute is reached, we first filter evaluation criteria mentioned fewer than five times (i.e., 1-4 times), and then select evaluation criteria which are mentioned 15+ times as *essential* evaluation criteria and the remaining ones as *optional* evaluation criteria. We take them as references for manual investigation. Detailed results will be presented in Section 4 (RQ1).

3.2 Investigating the Clarity of CRCs in Open-Source Projects

In this phase, we manually investigate the clarity of CRCs in open-source projects using the attributes and evaluation criteria of clarity derived in the prior phase.

3.2.1 Data Preparation. We use the benchmark dataset proposed by Li et al. [33] to conduct manual investigation. The dataset contains pairs of diff hunk and CRC written in nine programming languages. Specifically, we randomly sample a set of data from its validation dataset for each programming language. We do not sample from its training dataset because the data in different programming language is combined together and can not be distinguished. For each programming language, we randomly sample a set of data based on 95% confidence level and 5% confidence interval [10]. Table 2 presents the details of our sampled datasets. In total, we randomly sample 2,438 pairs of diff hunk and CRC. The sample size of each programming language varies from 216 for C to 339 for Golang.

3.2.2 Manual Investigation. Two authors of this paper first carefully read the attributes and evaluation criteria derived in the previous phase, and discuss until the two authors have a clear and consistent understanding on the details. For each sampled data, the two authors then independently examine the CRC and its corresponding code change to label if it meets the evaluation criteria for each attribute. When the process of labelling is completed, the two authors compare their results and discuss each disagreement until reaching a consensus. We have a Cohen's Kappa [41] value of 0.87 in this process, which indicates a substantial agreement. We will discuss the results of our manual investigate in Section 4 (RQ2).

Table 2. An overview of our studied open-source dataset.

Language	Original #	Sampled #
C	492	216
C++	736	253
C#	682	246
Golang	2,826	339
Java	1,636	312
JavaScript	1,035	281
PHP	443	206
Python	1,420	303
Ruby	1,049	282
Total	10,319	2,438

3.3 ClearCRC: Automatically Evaluating the Clarity of CRCs

In this phase, we propose ClearCRC, an automated framework that aims at the evaluation of the clarity of CRCs, based on the RIE attributes derived from our literature review and practitioners' feedback. We adopt different sets of backbone models in our framework to empirically study their effectiveness in automatically evaluating the clarity of CRCs.

3.3.1 Models. We use three sets of backbone models, including deep learning and machine learning models, pre-trained language models (e.g., CodeBERT [22] and CodeReviewer [33]), and large language models (e.g., Llama [56] and CodeLlama [50]).

Model Set 1: Deep Learning and Machine Learning Models. Prior studies on classifying good commit messages [55] and log messages [31] indicate that Bi-LSTM and Random Forest are effective in such classifications. Following these studies, we use Bi-LSTM [52] and Random Forest [12] as the deep learning and machine learning based backbones to perform the evaluation of CRCs' clarity.

Model Set 2: Pre-trained Language Models. For pre-trained language models, we use CodeBERT [22] and CodeReviewer [34] as our subject techniques. CodeBERT is a bimodal pre-trained language model for programming languages and natural languages with the same model architecture as RoBERTa-base [38]. It has been widely used by prior studies for classification tasks and presents a promising balance between performance and cost of computing resources [59, 66, 67]. CodeReviewer is a pre-trained model specialized for the automation of code review activities. They proposed pre-training tasks designed for code review like code diff denoising, and then pre-trained the CodeT5 [61] model on a large-scale code review dataset. CodeReviewer shows competitive performance on code review tasks such as code refinement.

Model Set 3: Large Language Models. LLMs have demonstrated promising results in various software engineering tasks [13, 15, 37], which brings opportunities and challenges for using LLMs as evaluators [14, 36]. For LLM baselines, we use *Llama3-70B-Instruct* [56] and *CodeLlama-34B-Instruct* [50]. We choose them since Llama series models show great performance among different LLMs [50, 56], and they are very popular in research related to code review [40, 64, 68]. Additionally, we also attempt to include a code-review-specialized LLM named LLaMA-Reviewer [40]. However, since LLaMA-Reviewer is tailored to specific downstream tasks and does not generalize well to our setting (i.e., it tends to generate invalid outputs when prompted), we finally decide to exclude it.

3.3.2 Data. Here we introduce the datasets we use as well as their augmentation and pre-processing.

Datasets and Augmentation. We utilize the manually labelled datasets in the prior phase to conduct the study, which consists of 2,438 pairs of code change and CRC in total. We randomly split the datasets into 80% training, 10% validation, and 10% testing. As shown in Table 4, the distribution of negative and positive instances is imbalanced in the dataset. To mitigate such impact, for each experiment, we perform up-sampling on the corresponding attribute. Specifically, we randomly repeat the negative instances of the experimented attribute in the training dataset to have the same amount as the positive instances. Note that we only augment data in the training set and ensure the testing set is consistent for all backbone models.

Processing. We analyze the raw input data including pairs of code change and CRC, process and combine the data with code change and CRC to feed into the model. We remove the lines of code that are unrelated to the code changes (e.g., the surrounding code of code changes). For models in set 1&2, we replace the *"-"* and *"+"* mark at the start of each line with *"[DELETE]"* and *"[ADD]"* in the code change, respectively. We then concatenate the CRC and the processed code change together, and attach a *"[SEP]"* token between them. For large language models in set 3, We embed the information of code change and CRC into the prompt and inference the models to obtain the results returned by the models and further evaluate the clarity of each attribute. For the prompt of using LLMs, we follow Prompt Engineering Guide to design the prompts [6]. As shown in Figure 5, we first provide an instruction of the task, the attributes, and evaluation criteria. We then inform the models of how to use the evaluation criteria (i.e., meet all of the essential ones and at least one of the optional ones) and the expected template of output. Finally, we attach the actual data (i.e., diff hunk and CRC) and have the model start its evaluation.

3.3.3 Evaluation. We introduce the evaluation details in our empirical study.

Metrics. We use four metrics to evaluate the results of ClearCRC and the baselines: 1) balanced accuracy, 2) precision, 3) recall, and 4) F-1 score. Balanced accuracy is computed based on the average of true positive rate and true negative rate. A higher balanced accuracy indicates a better capability in identifying both positive and negative instances. The balanced accuracy of random guess in binary classification is close to 50.0% [32]. It is widely used by prior work to evaluate the performance of binary classification, especially on imbalanced data [32, 69]. For the calculation of precision, recall, and F-1 score which focus on the classification performance of *positive* instances, we consider CRCs that meet the evaluation criteria as *positive* instances, and otherwise as *negative* instances.

K-Fold Cross Validation. We utilize 5-fold cross validation to mitigate the impact of randomness, where 5 is a commonly used K value in prior studies involving k-fold cross validation [24, 28, 43]. We randomly split the dataset into five subsets. The validation has five rounds in total. For each round of validation, we use one subset (i.e., 20%) for validation and testing (i.e., half for validation and half for testing), and the remaining four subsets (i.e., 80%) for training. We ensure that the dataset for each fold is identical for all models to perform a fair comparison.

3.3.4 Implementation Details. (1) For models in set 1, we use PyTorch [2] to implement Bi-LSTM and use Scikit-learn [3] to implement Random Forests, respectively. We follow prior studies [31, 55] to set the hyperparameters of the networks and the training processes. (2) For pre-trained models in set 2, we access the models through the official checkpoints released on HuggingFace [5]. As to hyperparameters like batch size and learning rate, we tune them according to the official replication package and the volume of our datasets. During the training stage, we set the number of training epochs to 10 and perform the strategy of early stopping (n=3) on all models to limit the training consumption and save the best-performing model on the validation dataset for further testing. We adopt the AdamW [39] optimizer and linear scheduler. (3) For large language models, we download

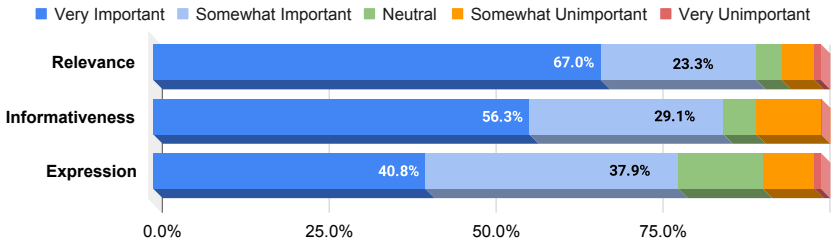


Fig. 3. Survey participants' rating for each of the attributes (RQ1).

their official checkpoints from HuggingFace. We set the maximum number of generated tokens to 32 which is appropriate for the output format, and keep the other parameters the same in the default configuration. We publicly release the scripts, parameters, and datasets for further research [1].

4 Results

In this section, we discuss the results of our RQs.

4.1 RQ1: Characterizing and Understanding the Clarity of CRCs

In this RQ, we discuss the results of our RIE attributes and evaluation criteria of CRCs' clarity, derived from our literature review and practitioners' survey. Table 3 shows an overview of the attributes and their evaluation criteria. Below, for each attribute, we discuss its detailed evaluation criteria and our survey results.

4.1.1 Relevance. If the code review comment is relevant to the code change.

Evaluation Criteria. After data analysis of open questions in our survey, we derive one essential (i.e., R.E1) and two optional (i.e., R.O1 and R.O2) evaluation criteria corresponding to the attribute of relevance. Figure 4 shows the frequency of evaluation criteria mentioned by our survey participants.

- R.E1 **Relevant to the code change.** The CRC should be self-explanatory and relevant to the code change. It can be interpreted based on the current code change without a relying relevance to external information (e.g., other CRCs).
- R.O1 **Specify the relevant location.** The CRC specifies the particular position of the code which has the issues or concerns.
- R.O2 **Correctly understand the code change.** The CRC explicitly shows that the reviewer correctly understands the code change.

Discussion. Figure 3 shows the percentage of each rate given by the survey participants regarding the importance of each attribute. Overall, most of the participants consider *Relevance* is important to the clarity of CRCs, including 67.0% as *very important* and 23.3% as *somewhat important*. Below, we present the comments from our survey participants regarding their perspectives on the evaluation criteria of each attribute. We correspond such comments to the evaluation criteria discussed above, and the corresponding part is highlighted in bold.

- R.E1 *"If the comment is made about the **intent or the change itself, not the state of the code in general**."*
- R.O1 *"Comments that do not pertain to the change as a whole, should refer directly to the **code elements** that should be modified in order for approval (e.g., **variable name, line of code**)."*
- R.O2 *"A relevant comment is one that is specific to the change and **shows a deep understanding of the code**."*

Table 3. An overview of the RIE attributes and their evaluation criteria derived from the survey.

Attribute	ID	Type	Description
Rel.	**R.E1**	Essential	Relevant to the code change.
	R.O1	Optional	Specify the relevant location.
	R.O2	Optional	Correctly understand the code change.
Info.	**I.E1**	Essential	Clear intention.
	I.E2	Essential	Provide reason or context information.
	I.O1	Optional	Provide suggestions for the next step.
	I.O2	Optional	Provide reference information.
Exp.	**E.E1**	Essential	Concise and to-the-point.
	E.E2	Essential	Polite and objective.
	E.O1	Optional	Readable format.
	E.O2	Optional	Proper syntax and grammar.

4.1.2 Informativeness. If the code review comment provides sufficient information.

Evaluation Criteria. Similar to the process discussed in the evaluation criteria of *Relevance*, there are 2 essential and 2 optional evaluation criteria corresponding to the attribute of *Informativeness*.

- I.E1 **Clear intention.** The CRC clearly specifies its intention (i.e., what is the further action needed) to make sure the CRC is actionable. The intention can include: 1) raising a question and asking for an answer; 2) identifying a problem that should be fixed; 3) providing suggestions that may be non-blocking and not urgent to take action.
- I.E2 **Provide reason or context information.** Based on the intention, provide context in the CRC. For example, (1) questioning: specifying what is the point of the question (e.g., not just "*Why?*"); (2) identifying issues: explaining what is the problem; (3) providing suggestions: the reason of such suggestions.
- I.O1 **Provide suggestions for the next step.** Try to provide suggestions for the next step if available.
- I.O2 **Provide reference information.** The CRC provides reference information that might be helpful to the target developer; such information may include the link to reference documents, guidelines, code, etc.

Discussion. As shown in Figure 3, over 85% of the participants consider that *Informativeness* is important to the clarity of CRCs. The remaining participants consider its importance as neutral or somewhat unimportant. However, the participants do not leave comments regarding the potential reasons. Below, we present the comments from our survey participants for their suggested evaluation criteria on *Informativeness*. Their comments are corresponded to the evaluation criteria discussed above and the related part is marked in bold.

- I.E1 *"It should be immediately obvious after reading the comment* ***what the commenter wants me to do****, why, and why their version is better than my version of the change." "One of the most important aspects to me is if the* ***intent of the comment is clear****."*
- I.E2 *"Including* ***reason or context*** *of why the code review comment is made." "Whether the comment contains a* ***reasoning for why*** *the change is wrong and needs to be amended."*
- I.O1 *"if the comment is rejecting a change, it should at least include a* ***suggestion of an alternative approach****."*
- I.O2 *"****Pointers and references to the materials*** *and existing discussions in the wild are important."*

4.1.3 Expression. If the code review comment is readable, easy to understand, and friendly.

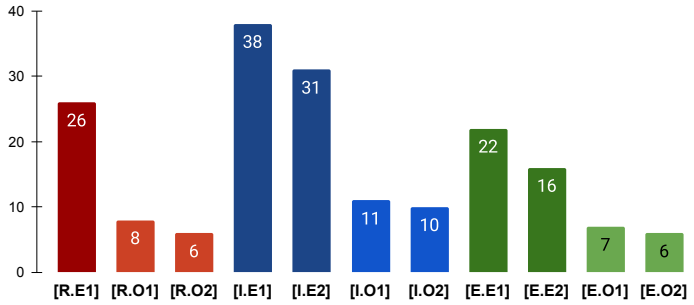


Fig. 4. Frequency of each evaluation criteria mentioned in the survey.

Evaluation Criteria. There are 2 essential and 2 optional evaluation criteria corresponding to the attribute of *Expression*.

- E.E1 **Concise and to-the-point.** Describe the idea as precise and concise as possible to avoid vagueness, ambiguity, and incoherence.
- E.E2 **Polite and objective.** The CRC should express the idea in a polite manner, and focus on the code rather than the person.
- E.O1 **Readable format.** The CRC is written in a human readable format.
- E.O2 **Proper syntax and grammar.** The CRC is written in a correct syntax and grammar, without typos or incomplete words.

Discussion. As shown in Figure 3, 78.7% (i.e., 40.8% *very important* + 37.9% *somewhat important*) of the survey participants acknowledge the importance of *Expression* to the CRC's clarity. There are 12.6% of the participants consider its importance as *neutral* and 8.8% as *unimportant*. For example, one participant that selects *neutral* comments *"It dependes on what the expression be used for. A long but easy to understant expression is ok"*. One participant that selects *somewhat unimportant* comments *"Really depends on the working relations between the developer and the reviewer. As long as they can understand each-other all is well"*. Overall, *Expression* has a relatively lower positive rate compared to the other two attributes, but still accounts for the majority of the participants.

Below, we present the comments regarding the evaluation criteria suggested by our survey participants who acknowledge the importance of *Expression*.

- E.E1 *"When evaluating the expression of a code review comment, you're looking at how well the feedback is communicated, whether it is clear,* ***concise, and effectively conveys*** *the reviewer's thoughts"*
- E.E2 *"Comments should have* ***friendly tone and comment on the code, not the person****."*
- E.O1 *"****Formatting around non-english or code snippets****. (i.e. backticks "). This helps improve overall clarity."*
- E.O2 *"Comment should be plain human readable sentences, because PR author and other reviewers are humans.* ***Proper syntax and grammar, absence of typos*** *are important as well."*

4.1.4 Practitioners' Feedback on Additional Attributes. Apart from the three attributes, we also ask the participants for additional aspects or attributes that may contribute to the clarity of CRCs. In total, we receive 48 responses to this question. After removing three responses that are *"N/A"* or *"None"*, we analyze the remaining 45 responses and summarize them as follows. Note that a response may be summarized into multiple aspects.

- **Consistent to our attributes or evaluation criteria.** We find that the comments of 31 participants are consistent to our attributes or the evaluation criteria. For example, *"Providing references (links to other discussions, code changes, documentation, etc.) is important for building trust and minimizing the length of feedback cycles"* is consistent to I.O2 and *"Politeness"* is consistent to E.E2.
- **Overall expectations on CRCs.** There are 11 participants comment their expectations on CRCs, which may not be directly related to clarity. For example, *"The reviewer should be responsive i.e. should quickly respond to the questions raised by the contributor"* is about the time taken to reply and *"Some reviews will require a history to develop how experienced the reviewee is so the appropriate level of explanation is given for their skill"* is about writing CRCs based on reviewee's knowledge.
- **Other suggestions on code review.** There are 8 participants comment on other suggestions regarding the practice of code review. For example, *"Rich features of the code review tool is also important. For example, GitHub provides a "suggestion" feature, it makes the suggestion of the code change clear"* is about leveraging tools to provide suggestions and *"I'm not sure if this it too meta, but I think having a space to talk about what kind of culture you want to encourage is important to setting standards"* is about the role of code review in building relationship and culture.

Overall, there is no additional attribute derived from the practitioners' feedback on potential new attributes. However, they provide valuable insights of their expectations on code review, and may inspire future studies to improve the quality of CRCs and the practice of code review.

Summary of RQ1: Based on our literature review and survey with open-source practitioners, we derive three attributes related to the clarity of CRCs and their corresponding evaluation criteria. A majority of the practitioners consider these three attributes important to the clarity of CRCs.

4.2 RQ2: Clarity of CRCs in Open-Source Projects

In this RQ, we first present our detailed process on analyzing the results of our manual investigation. We then present the results of our quantitative analysis and case study, respectively.

4.2.1 Experimental Setup. Two authors of this paper independently label the clarity of CRCs following the process and using the datasets discussed in Section 3. Particularly, for each attribute, the CRC will be marked as *positive* if it meets *all* of the essential criteria and *at least one* of the optional criteria. Otherwise, it will be marked as *negative*. During the data annotation, each attribute is separately and independently labelled, and the results for one indicator will not affect those of the other two attributes. When the labelling is completed, the two authors compare their results and discuss each disagreement until reaching a consensus. The value of Cohen's Kappa [41] in this process is 0.87, which indicates a substantial agreement.

4.2.2 Quantitative Analysis. We present the results of our quantitative analysis on the clarity of CRCs by different programming languages. Table 4 shows the percentage of CRCs' clarity for each programming language. Specifically, *"Negative"* refers to the percentage of CRCs that do not meet the evaluation criteria for each attribute, *"All positive"* refers to the percentage of CRCs that meet the evaluation criteria for all the three attributes. Overall, 71.2% of the CRCs meet the evaluation criteria in all of the three attributes, meaning that a large portion of the CRCs (i.e., 28.8%) is not shown to have a sufficient clarity. We discuss the results by comparing among the attributes and the programming languages, respectively.

Comparison among attributes. The distribution of CRCs that are negative for different attributes is 11.4% on average for *Relevance*, 19.3% on average for *Informativeness*, and 5.8% on average for

Expression, respectively. The results show that a non-negligible portion of CRCs in open-source projects is not written with good clarity, especially for *Informativeness* and *Relevance*.

Comparison among programming languages. We find that the distribution of clarity varies for different programming languages. For example, over 75% of the CRCs for *C* and *Java* are all positive. Differently, only 63.6% of the CRCs are all positive for *C++*, meaning that over 35% of its CRCs have an insufficient clarity.

4.2.3 Case Study. For each attribute, we discuss a negative example (i.e., does not meet the evaluation criteria discussed in the experimental setup of this RQ) and a positive example (i.e., meets the evaluation criteria), respectively. Note that we rename the identifier names and slightly rephrase the CRC to avoid directly retrieving the author of the CRC based on provided samples.

Relevance. As presented in the examples below, the negative example comments *"Same here. and also all others"*. The CRC itself hardly contains any information relevant to the code change. It may only be relevant to the information outside this code change. Therefore, it is not shown to be relevant to the code change, and it is not self-interpretable. Therefore, this CRC does not meet the essential criteria of *R.E1*. In comparison, the positive example raises a question, and the question is shown to be relevant to the code change. Moreover, it specifies the exact location in the code change where the reviewer has a question.

```
# Negative Example (Ruby)
+    def print_the_page(**options)
+      options[:page_ranges] &&= Array(options[:page_ranges])
+      bridge.print_the_page(options)
+    end

CRC: Same here. and also all others.

# Positive Example (Python)
-    self._internal = self._internal.resolved_copy
+    self._update_internal_frame(
+        self._internal.resolved_copy, requires_same_anchor=False

CRC: When do we need to set 'requires_same_anchor=False?'
```

Informativeness. As shown in the negative example below, the CRC mentions "*This change is not correct*". This CRC may imply the developer to revert this change or fix an issue. However, the comment does not provide an explanation of why the change is considered incorrect. It's important to offer specific reasons or context information to help the developer understand the issue. Therefore, this example CRC does not meet the essential criteria of *I.E2*. In comparison, the positive example explains the issue and further provides a suggestion for the further action to reproduce the problem.

```
# Negative Example (JavaScript)
       hosts.add(host);
-      LOG.info(String.format(""Added node %s."", node.getId()));
+      LOG.finest(String.format(""Added node %s."", node.getId()));
       host.runHealthCheck();

CRC: This change is not correct.

# Positive Example (JavaScript)
+       process.on('SIGUSR2', function () {
+               log.reopenFileStreams();
+       });
+
         module.exports.logger = logger;

CRC: Check the linting is failing, 'log' is not defined.
You can run locally 'npm run lint' to double check.
```

Table 4. Distribution (%) of CRCs' clarity for each programming language (RQ2).

Language	Negative			All Positive
	Relevance	Informativeness	Expression	
C	11.1	14.4	5.1	77.3
C++	11.5	28.1	6.3	63.6
C#	9.8	25.6	8.9	63.8
Golang	11.2	16.2	3.8	74.0
Java	6.7	17.9	6.1	75.3
JavaScript	9.3	17.1	5.3	72.2
PHP	15.0	15.5	3.4	70.4
Python	11.2	18.2	4.0	73.6
Ruby	18.4	20.9	9.6	68.8
Overall	11.4	19.3	5.8	71.2

Expression. The CRC in the negative example asks the question in an impolite manner, which does not meet the essential evaluation criteria of *E.E2*. According to many of our survey participants' feedback, being polite and friendly is very important to efficient communications. Constructive criticism and polite suggestions for improvement are always preferred than harsh or toxic comments.

```
# Negative Example (C#)
-       (i1, s2, err2, s2) =>
+       (i1, s2, errCode, err2, s2) =>

CRC: wtf is i1, s2, errCode, err2, s2?
I know we have nested lambdas by maybe this is a case for a method.

# Positive Example (JavaScript)
 using Telemetry;
+using Telemetry.Trace;

CRC: This is technically correct, I wonder if we could make it
simpler by not requiring this namespace?
```

Summary of RQ2: We find that a large portion (i.e., 28.8%) of the CRCs in our study open-source datasets have insufficient clarity. Among the three attributes, *Informativeness* has the most noticeable insufficiency.

4.3 RQ3: Automatically Evaluating the Clarity of CRCs

In this RQ, we present the results of our evaluation on the clarity of CRCs.

4.3.1 Main Results. Table 5 presents the results of ClearCRC, organized by different RIE attributes, model, and metrics. In the table, we report the average results of five-fold cross validation. The bold number of each column shows the best performance of the corresponding attribute and metric.

Comparison among model sets. Overall, pre-trained language models achieve the best performance among all model sets. Specifically, with regard to the set average performance across all evaluation attributes, pre-trained language models have the best performance on all four metrics. For example, the balanced accuracy of set 2 models is 71.25%, outperforming other sets by a large margin (i.e., 17.7% and 20.8% relative improvements compared to set 1 and set 3 models, respectively). Apart from balanced accuracy, there is also a consistent advantage for set 2 models on the other three metrics. We believe that the strong performance of pre-trained language models is attributed to their prior code knowledge and task-specific fine-tuning. In contrast, large language models perform poorly due to their inability to acquire sufficient knowledge about the clarity of CRCs.

Instructions
You are a powerful model in evaluating the clarity of code review comments. Your task is to evaluate the clarity of code review comments based on the following attributes:

Relevance: (1) Relevant to the code change. (2) Specify the relevant location. (3) Correctly understand the code change. Mark relevance as "1" if it meets (1) and one of (2) (3), other wise mark as "0"

Informativeness: (1) Clear intention. (2) Provide context information. (3) Provide suggestions for the next step. (4) Provide reference information. Mark informativeness as "1" if it meets (1) (2) and one of (3) (4), other wise mark as "0"

Expression: (1) Concise and to-the-point. (2) Polite and objective. (3) Readable format. (4) Proper syntax and grammar. expression as "1" if it meets (1) (2) and one of (3) (4), other wise mark as "0".

Format of Output
Below is the format of an example response:

Relevance: 1/0
Informativeness: 1/0
Expression: 1/0

Task
Now it's your turn, you only need to output the labels after [output] based on [patch] and [code review comment]. Don't output any additional contents.
[patch]
{code}

[code review comment]
{msg}

[output]

Fig. 5. Prompt template for using LLMs to evaluate the clarity of CRCs.

Comparison among attributes. We find that the models' performance on different attributes varies a lot. The average balanced accuracy of *Informativeness* is 71.30%, while the number is 56.91% for *Expression*, which may suggest that *Expression* is a relatively easier attribute for subject models to understand and recognize, but harder for *Informativeness*. However, we also notice that the recall of *Informativeness* is much lower than *Relevance* and *Expression* (i.e., 78.22% compared to 83.41% and 95.87%). We conjecture that since the dataset of *Informativeness* has a higher proportion of negative samples, the model achieves a higher balanced accuracy in distinguishing positive and negative samples while also making it more difficult to recall negative samples.

Comparison within the set. We mainly analyze the results in set 2 models because they are the best set among all models. For average performance among all attributes, CodeBERT is better in terms of balanced accuracy (i.e., 73.04% > 69.46%) but behind CodeReviewer on F-1 score (i.e., 93.07% > 94.61%). Overall, the two models have comparable strengths. Considering that the size of CodeBERT (i.e., 125M) is about half of CodeReviewer's (i.e., 223M), we believe CodeBERT demonstrates good generalizability for automatically evaluating CRC's clarity.

Overall, the results show that in terms of precision, eecall, and F-1 score, the performance of these models is satisfactory (i.e., an overall average score of more than 85%), but there is still room for improvement for balanced accuracy (i.e., 63.58%), which could be credited to the suboptimal ability to detect negative CRC samples. Therefore, we believe that ClearCRC is promising in automatically evaluating the clarity of CRCs and further exploration is still required.

4.3.2 Generalizability On Other Datasets. To evaluate if ClearCRC could generalize to newer or less-studied projects, we conduct a study on a subset of CodeReviewer-New [25], which includes repositories that the original CodeReviewer dataset does not contain, and adopts various approaches

Table 5. Balanced Accuracy (BA, %), Precision (P, %), Recall (R, %), and F-1 score (%) of ClearCRC using different models. **Bold** value: Maximum value in each metric of each attribute.

Models	Relevance				Informativeness				Expression				Average			
	BA	P	R	F1	BA	P	R	F1	BA	P	R	F1	BA	P	R	F1
Set 1: Machine Learning and Deep Learing Models																
Random Forest	52.77	88.04	**99.91**	93.60	69.90	89.19	87.60	88.38	53.13	94.73	**99.82**	97.21	58.60	90.66	95.78	93.06
LSTM	58.66	89.58	95.50	92.39	68.93	90.70	72.49	80.19	59.78	95.48	96.72	96.08	62.45	91.92	88.24	89.55
Set 1 Avg.	*55.72*	*88.81*	*97.70*	*92.99*	*69.41*	*89.95*	*80.04*	*84.28*	*56.45*	*95.11*	*98.27*	*96.64*	*60.53*	*91.29*	*92.01*	*91.31*
Set 2: Pre-trained Language Models																
CodeBERT	**78.37**	**95.12**	88.48	91.27	77.43	92.22	88.78	90.42	**63.31**	**95.89**	99.22	**97.53**	**73.04**	**94.41**	92.16	93.07
CodeReviewer	76.83	93.91	97.30	**95.55**	73.02	90.02	**92.95**	**91.41**	58.53	95.39	98.44	96.87	69.46	93.11	**96.23**	**94.61**
Set 2 Avg.	*77.60*	*94.51*	*92.89*	*93.41*	*75.23*	*91.12*	*90.86*	*90.92*	*60.92*	*95.64*	*98.83*	*97.20*	*71.25*	*93.76*	*94.19*	*93.84*
Set 3: Large Language Models																
Llama	61.76	90.42	86.90	88.60	**78.62**	**94.12**	79.68	86.30	52.14	94.58	98.16	96.90	64.17	93.04	88.25	90.60
CodeLlama	46.74	85.16	32.36	46.90	59.92	88.72	47.82	62.12	54.56	94.94	82.84	88.46	53.74	89.61	54.34	65.83
Set 3 Avg.	54.25	87.79	59.63	67.75	69.27	91.42	63.75	74.21	53.35	94.76	90.50	92.68	58.96	91.32	71.29	78.21
Overall																
Average	62.52	90.37	83.41	84.72	71.30	90.83	78.22	83.14	56.91	95.17	95.87	95.51	63.58	92.12	85.83	87.79

Table 6. Results on CodeReviewer-New for set 2 models.

Models	Relevance				Informativeness				Expression				Average			
	BA	P	R	F1	BA	P	R	F1	BA	P	R	F1	BA	P	R	F1
CodeBERT	61.39	91.94	94.21	93.06	63.27	74.77	87.91	80.81	75.45	88.12	87.25	87.68	66.70	84.94	89.79	87.18
CodeReviewer	64.55	92.62	93.39	93.00	63.67	74.17	97.80	84.36	80.48	90.91	88.24	89.55	69.57	85.90	93.14	88.97
Average	62.97	92.28	93.80	93.03	63.47	74.47	92.86	82.59	77.97	89.52	87.75	88.62	68.14	85.42	91.47	88.08

to ensure the data quality. We randomly sample 135 examples from CodeReviewer-New (i.e., 15 samples for each of the 9 languages), and follow the same data annotation approaches and experimental settings as the main experiments. We use the best checkpoint in the fold 1 cross validation of the main experiments for each model.

Compared to the results with the original CodeReviewer dataset, there is a slight drop of the performance for both models. For example, the balanced ccuracy decreases by around 3% (i.e., 71.25% -> 68.14%) and the F-1 score decreases by 5% (i.e., 93.84% -> 88.08%). Considering the different time and project distributions of the two datasets, we think that the decline is still reasonable and ClearCRC could be generalized to the additional datasets.

Summary of RQ3: ClearCRC assisted with pre-trained language models shows promising results for automatic evaluation of the clarity of CRCs, with a balanced accuracy and F-1 score of 71.25% and 93.84%, respectively. Additionally, it could be generalized to newer or less-studied datasets.

5 Discussion

5.1 Implications

5.1.1 Implication 1: Actionable guidelines for evaluating and writing clear CRCs. As shown in our results of RQ2, there is a large portion of the CRCs in open-source systems (i.e., 28.8%) that lack clarity. Due to the lack of well-defined guidelines on writing CRCs, it is challenging for developers to write CRCs that can clearly and sufficiently serve as the medium between developers and reviewers. Based on our survey with open-source practitioners, many participants mention that they do not

want to see CRCs that are *"confusing"* and *"vague"*. Instead, they expect the CRCs to be *"clear"*. However, it is also difficult to determine what is a *"clear"* CRC.

In our study, we characterize the clarity of CRCs and derive three attributes with their corresponding evaluation criteria. One of our survey participants mentions *"I like the idea of thinking more about comments. I would welcome good guidelines."*. Therefore, our findings can be used as actionable guidelines for evaluating and writing clear CRCs, which in turn improves the efficiency and quality of the code review process.

5.1.2 Implication 2: Select high-quality data for the automated generation of CRCs. There are a series of studies that utilize existing CRC data to train models for the automated generation of CRCs [33, 40, 57]. However, these studies accept all the CRC data in general, without a curation or selection on the quality of data. Based on such a situation, existing CRC generation techniques may learn from CRC data with insufficient clarity and then generate confusing results.

In our survey, we also ask the participants for their opinion on automated CRC generation techniques. They can rate the importance of the clarity of automatically generated CRCs from 1 to 5, where 1 indicates the lowest importance and 5 indicates the hightest importance. Figure 6 presents the results of their ratings. The average rating is 4.04, and more than 70% of the participants consider its importance as 4 or 5. For example, one participant comments that *"If I am going to receive automated comments on my code changes, they need to be clear, accurate, and relevant. Otherwise they are just wasting my time"*. As shown in the results of RQ3, ClearCRC achieves a high precision in evaluating the clarity of CRCs on all the three attributes. Therefore, the findings of our study can be used for implementing data filtering and selection mechanisms to help identify CRCs with good clarity, improving the overall effectiveness of the automated CRC generation process.

5.1.3 Implication 3: Provide a more comprehensive quality evaluation for CRCs and its generation. As discussed above, developers expect to have CRCs with good quality to foster an effective communication among the team members. Moreover, existing research on automated CRC generation generally uses BLEU score [44] to examine the textual similarity between the generated CRC and the reference CRC. While BLEU score can be used to assess the performance of automated CRC generation techniques, it does not directly address the quality of the CRC itself. In other words, a high Bleu score does not necessarily indicate that the generated CRC is clear and concise. Therefore, the quality of the CRC itself still remains unclear.

In this paper, we study the quality of CRC by understanding and uncovering the clarity of CRC. To do so, we derive the RIE attributes (i.e., Relevance, Informativeness, and Expressiveness) and their respective evaluation criteria. These attributes and criteria can be leveraged to provide a more comprehensive evaluation for CRCs and their automated generation. By utilizing our findings, we aim to provide a more comprehensive quality evaluation for CRCs and their automated generation. It can help developers in writing better CRCs, and also contribute to the improvement of automated CRC generation techniques, ultimately leading to more effective communication among software development teams.

5.2 RIE Indicators and Existing Metrics

5.2.1 Comparison with Existing Metrics. Despite the RIE attributes we derive from this paper, the community and research have proposed other metrics to evaluate the quality of CRCs such as Readability, Sentiment [7], and Usefulness [47]. Below, we compare these existing metrics with our RIE attributes:

- **Readability**: Readability is seen as an important quality dimension of software comments [21], and it emphasizes the difficulty of reading the text (e.g., the number of difficult words and length

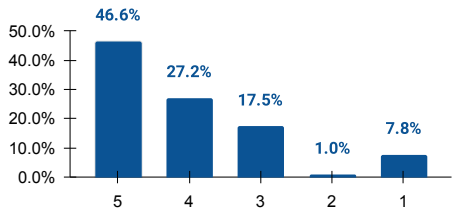


Fig. 6. Rating of importance for the clarity of generated CRC in our survey.

of the sentence). It is a subset of our Expression indicator (i.e., E.O1), and Expression includes other aspects like the tone. Besides, a readable CRC could easily be unclear. For example, a comment generated by LLM is typically very easy to read, but it may contain little valuable and helpful information for further action.

- **Sentiment** [7]: In code review activities, the contributors frequently express positive, neutral, and negative sentiments, and these sentiments are correlated with the complete time of review [7]. Sentiment is related to the evaluation criteria of Expression (i.e., E.E2 "Polite and objective"), as polite and objective comments are usually positive or neutral. However, sentiment alone is not enough to assess the clarity of CRC—purely encouraging and positive comments are not necessarily clear, while neutral CRC could achieve sufficient clarity.
- **Usefulness** [47]: Usefulness of one CRC is typically measured based on the outcome of the corresponding code change (e.g., acceptance rate and time) in former research [47], on the assumption that more and faster code changes are useful because they can lead to better software quality. However, the outcome of one code change depends on many other factors like the identity of the contributor, code style, and the change scope [48], which adds more indeterministic to this measure. In contrast to Usefulness, which could be deemed as an outcome-driven metric, the RIE attributes are driven by the process of code review activities, mainly focusing on the clarity of CRCs themselves.

In this paper, instead of introducing a single new metric, we present a set of well-defined and actionable evaluation criteria for assessing the clarity of CRCs. These criteria are derived from a systematic process and align with the shared expectations of practitioners across academia, industry, and the open-source community.

5.2.2 Relation Among RIE attributes. The RIE attributes aim to measure the clarity of CRCs from three distinct dimensions. They are conceptually orthogonal, meaning that each dimension assesses a distinct quality aspect and is relatively independent of the others:

- **Relevance**: *Relevance* primarily assesses the degree and correctness of the relevance between CRCs and code changes.
- **Informativeness**: *Informativeness* mainly evaluates whether one CRC provides useful information like intention, explanation, suggestion, and reference information.
- **Expression**: *Expression* measures if the CRC is expressed appropriately from perspectives like readability.

Since these dimensions capture different aspects, a CRC may perform well in one dimension but poorly in another. For example, a CRC may be highly relevant to code but lack meaningful information, or it may contain rich details but be poorly written and hard to understand. Because of this, the three dimensions should be evaluated separately to comprehensively assess the clarity of CRCs. Besides, we would like to mention that although the concepts and evaluation criteria are

independent, some attributes are indeed correlated and likely co-occur. For example, if one code reviewer is merely complaining about a specific code change, the comment is typically neither informative nor polite.

6 Threats to Validity

6.1 Internal Validity

We manually label the clarity of CRCs on open-source datasets. To mitigate the subjective bias in this process, two authors of this paper label the data independently. The labellers then discuss each disagreement until a consensus is reached. Following prior studies [18, 32, 35], we use Cohen's Kappa [41] to measure the agreement of the manual investigation results between the two authors. The Cohen's Kappa value in this process is 0.87, which indicates a substantial agreement. The randomness in the process of our experiments (e.g., splitting the data, training the models) may affect the results. To mitigate such threats, we use a five-fold cross validation to conduct the experiments and report the average number in our discussions. We derive the evaluation criteria by analyzing the 103 survey responses from participants. Engaging more experts from various domains may generate more comprehensive results.

6.2 External Validity

We conduct our study on the datasets proposed by Li et al. [33]. Using other datasets may generate different results and findings. However, the datasets of Li et al. [33] extract code changes and the corresponding CRCs from open-source projects written in nine programming languages, which include a diverse range of repositories. We derive the detailed evaluation criteria based on the survey with practitioners from open-source projects written in nine programming languages. However, the findings of our study are not specialized for specific programming languages and can be generalizable to various projects. Our study is conducted based on open-source data and practitioners. Future studies may verify the generalizability of our findings on industrial systems and projects.

7 Conclusion

In this paper, we investigate how a code review comment (CRC) can clearly and concisely serve as the medium of communication among developers by conducting a multi-phased, comprehensive study. We derive our RIE attributes of the clarity of CRCs and the detailed evaluation criteria based on the analysis of our literature review and survey with practitioners. We also find that a noticeable portion of the CRCs in open-source projects do not have sufficient clarity. We further seek to explore the potential of automatically evaluating the clarity of CRCs by proposing an automated framework, namely ClearCRC. Experimental results show that ClearCRC is effective in evaluating the clarity of CRCs based on our RIE attributes and outperforms the baseline approaches by a considerable margin. Our findings shed light on characterizing the quality of CRCs and further facilitate the collaboration between developers.

Data Availability

Our replication package is available and can be accessed using the link [1].

Acknowledgment

This work was supported by the National Key R&D Program of China (No. 2024YFB4506400), sponsored by CCF-Huawei Populus Grove Fund, and National Natural Science Foundation of China (No. 62172214).

References

[1] [n. d.]. Link to our replication package. https://github.com/iCSawyer/ClearCRC. Last accessed Apr 2025.
[2] [n. d.]. PyTorch. https://pytorch.org/. Last accessed October 2024.
[3] [n. d.]. scikit-learn: Machine Learning in Python. https://scikit-learn.org. Last accessed October 2024.
[4] 2023. Microsoft Forms. https://forms.office.com/. Last accessed October 2024.
[5] 2024. Hugging Face – The AI community building the future. https://huggingface.co/. Last accessed October 2024.
[6] 2024. Prompt Engineering Guide. https://www.promptingguide.ai/. Last accessed October 2024.
[7] Ikram El Asri, Noureddine Kerzazi, Gias Uddin, Foutse Khomh, and M.A. Janati Idrissi. 2019. An empirical study of sentiments in code reviews. *Information and Software Technology* 114 (2019), 37–54.
[8] Vipin Balachandran. 2013. 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)*. IEEE, 931–940.
[9] Moritz Beller, Alberto Bacchelli, Andy Zaidman, and Elmar Juergens. 2014. Modern code reviews in open-source projects: Which problems do they fix?. In *Proceedings of the 11th working conference on mining software repositories*. 202–211.
[10] S. Boslaugh and P.A. Watters. 2008. *Statistics in a Nutshell: A Desktop Quick Reference*. O'Reilly Media.
[11] Amiangshu Bosu, Michaela Greiler, and Christian Bird. 2015. Characteristics of useful code reviews: An empirical study at microsoft. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 146–156.
[12] Leo Breiman. 2001. Random forests. *Machine learning* (2001), 5–32.
[13] Junkai Chen, Xing Hu, Zhenhao Li, Cuiyun Gao, Xin Xia, and David Lo. 2024. Code search is all you need? improving code suggestions with code search. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE)*. 1–13.
[14] Junkai Chen, Zhenhao Li, Xing Hu, and Xin Xia. 2024. Nlperturbator: Studying the robustness of code llms to natural language variations. *arXiv preprint arXiv:2406.19783* (2024).
[15] Junkai Chen, Zhiyuan Pan, Xing Hu, Zhenhao Li, Ge Li, and Xin Xia. 2025. Reasoning Runtime Behavior of a Program with LLM: How Far Are We?. In *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering (ICSE)*.
[16] Moataz Chouchen, Ali Ouni, Raula Gaikovina Kula, Dong Wang, Patanamon Thongtanunam, Mohamed Wiem Mkaouer, and Kenichi Matsumoto. 2021. Anti-patterns in modern code review: Symptoms and prevalence. In *2021 IEEE international conference on software analysis, evolution and reengineering (SANER)*. IEEE, 531–535.
[17] Nicole Davila and Ingrid Nunes. 2021. A systematic literature review and taxonomy of modern code review. *Journal of Systems and Software* (2021).
[18] Zishuo Ding, Yiming Tang, Yang Li, Heng Li, and Weiyi Shang. 2023. On the temporal relations between logging and code. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 843–854.
[19] Felipe Ebert, Fernando Castor, Nicole Novielli, and Alexander Serebrenik. 2019. Confusion in code reviews: Reasons, impacts, and coping strategies. In *2019 IEEE 26th international conference on software analysis, evolution and reengineering (SANER)*. IEEE, 49–60.
[20] Felipe Ebert, Fernando Castor, Nicole Novielli, and Alexander Serebrenik. 2021. An exploratory study on confusion in code reviews. *Empirical Software Engineering* 26 (2021), 1–48.
[21] Derar Eleyan, Abed Othman, and Amna Eleyan. 2020. Enhancing software comments readability using flesch reading ease score. *Information* 11, 9 (2020), 430.
[22] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP*. Association for Computational Linguistics, 1536–1547.
[23] Isabella Ferreira, Jinghui Cheng, and Bram Adams. 2021. The" shut the f** k up" phenomenon: Characterizing incivility in open source code review discussions. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW2 (2021), 1–35.
[24] Ramin Ghorbani and Rouzbeh Ghousi. 2020. Comparing different resampling methods in predicting students' performance using machine learning techniques. *IEEE access* 8 (2020), 67899–67911.
[25] Qi Guo, Junming Cao, Xiaofei Xie, Shangqing Liu, Xiaohong Li, Bihuan Chen, and Xin Peng. 2024. Exploring the potential of chatgpt in automated code refinement: An empirical study. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–13.
[26] Anshul Gupta and Neel Sundaresan. 2018. Intelligent code reviews using deep learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'18) Deep Learning Day*.
[27] Masum Hasan, Anindya Iqbal, Mohammad Rafid Ul Islam, AJM Imtiajur Rahman, and Amiangshu Bosu. 2021. Using a balanced scorecard to identify opportunities to improve code review effectiveness: An industrial experience report. *Empirical Software Engineering* 26 (2021), 1–34.

[28] Ömer Karal. 2020. Performance comparison of different kernel functions in SVM for different k value in k-fold cross-validation. In *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*. IEEE, 1–5.
[29] Oleksii Kononenko, Olga Baysal, and Michael W. Godfrey. 2016. Code Review Quality: How Developers See It. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. 1028–1038.
[30] Oleksii Kononenko, Olga Baysal, Latifa Guerrouj, Yaxin Cao, and Michael W Godfrey. 2015. Investigating code review quality: Do people and participation matter?. In *2015 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 111–120.
[31] Zhenhao Li, An Ran Chen, Xing Hu, Xin Xia, Tse-Hsun Chen, and Weiyi Shang. 2023. Are They All Good? Studying Practitioners' Expectations on the Readability of Log Messages. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 129–140.
[32] Zhenhao Li, Tse-Hsun Chen, and Weiyi Shang. 2020. Where Shall We Log? Studying and Suggesting Logging Locations in Code Blocks. In *35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020*. 361–372.
[33] Zhiyu Li, Shuai Lu, Daya Guo, Nan Duan, Shailesh Jannu, Grant Jenks, Deep Majumder, Jared Green, Alexey Svyatkovskiy, Shengyu Fu, et al. 2022. Automating code review activities by large-scale pre-training. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1035–1047.
[34] Zhiyu Li, Shuai Lu, Daya Guo, Nan Duan, Shailesh Jannu, Grant Jenks, Deep Majumder, Jared Green, Alexey Svyatkovskiy, Shengyu Fu, et al. 2022. Automating code review activities by large-scale pre-training. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1035–1047.
[35] Zhenhao Li, Chuan Luo, Tse-Hsun Chen, Weiyi Shang, Shilin He, Qingwei Lin, and Dongmei Zhang. 2023. Did we miss something important? studying and exploring variable-aware log abstraction. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 830–842.
[36] Zongjie Li, Chaozheng Wang, Pingchuan Ma, Daoyuan Wu, Shuai Wang, Cuiyun Gao, and Yang Liu. 2024. Split and Merge: Aligning Position Biases in LLM-based Evaluators. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. 11084–11108.
[37] Junwei Liu, Kaixin Wang, Yixuan Chen, Xin Peng, Zhenpeng Chen, Lingming Zhang, and Yiling Lou. 2024. Large language model-based agents for software engineering: A survey. *arXiv preprint arXiv:2409.02977* (2024).
[38] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).
[39] Ilya Loshchilov and Frank Hutter. [n. d.]. Decoupled Weight Decay Regularization. ([n. d.]).
[40] Junyi Lu, Lei Yu, Xiaojia Li, Li Yang, and Chun Zuo. 2023. LLaMA-Reviewer: Advancing Code Review Automation with Large Language Models through Parameter-Efficient Fine-Tuning. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 647–658.
[41] Mary L. McHugh. 2012. Interrater reliability: the kappa statistic. *Biochemia Medica* 22, 3 (2012), 276–282.
[42] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. 2014. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In *Proceedings of the 11th working conference on mining software repositories*. 192–201.
[43] Isaac Kofi Nti, Owusu Nyarko-Boateng, Justice Aning, et al. 2021. Performance of machine learning algorithms with different K values in K-fold CrossValidation. *International Journal of Information Technology and Computer Science* (2021), 61–71.
[44] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 311–318.
[45] Luca Pascarella, Davide Spadini, Fabio Palomba, Magiel Bruntink, and Alberto Bacchelli. 2018. Information needs in contemporary code review. *Proceedings of the ACM on human-computer interaction* 2, CSCW (2018), 1–27.
[46] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research* 21, 140 (2020), 1–67.
[47] Mohammad Masudur Rahman, Chanchal K. Roy, and Raula G. Kula. 2017. Predicting Usefulness of Code Review Comments Using Textual Features and Developer Experience. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. 215–226.
[48] Achyudh Ram, Anand Ashok Sawant, Marco Castelluccio, and Alberto Bacchelli. 2018. What makes a code change easier to review: an empirical investigation on code change reviewability. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 201–212.

[49] Peter Rigby, Brendan Cleary, Frederic Painchaud, Margaret-Anne Storey, and Daniel German. 2012. Contemporary Peer Review in Action: Lessons from Open Source Development. *IEEE Software* 29, 6 (2012), 56–61.
[50] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950* (2023).
[51] Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern code review: a case study at google. In *Proceedings of the 40th international conference on software engineering: Software engineering in practice*. 181–190.
[52] Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing* (1997), 2673–2681.
[53] Jing Kai Siow, Cuiyun Gao, Lingling Fan, Sen Chen, and Yang Liu. 2020. Core: Automating review recommendation for code changes. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 284–295.
[54] Donna Spencer. 2009. *Card sorting: Designing usable categories*. Rosenfeld Media.
[55] Yingchen Tian, Yuxia Zhang, Klaas-Jan Stol, Lin Jiang, and Hui Liu. 2022. What makes a good commit message?. In *Proceedings of the 44th International Conference on Software Engineering*. 2389–2401.
[56] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).
[57] Rosalia Tufano, Simone Masiero, Antonio Mastropaolo, Luca Pascarella, Denys Poshyvanyk, and Gabriele Bavota. 2022. Using pre-trained models to boost code review automation. In *Proceedings of the 44th international conference on software engineering*. 2291–2302.
[58] Rosalia Tufano, Luca Pascarella, Michele Tufano, Denys Poshyvanyk, and Gabriele Bavota. 2021. Towards automating code review activities. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 163–174.
[59] Asif Kamal Turzo, Fahim Faysal, Ovi Poddar, Jaydeb Sarker, Anindya Iqbal, and Amiangshu Bosu. 2023. Towards Automated Classification of Code Review Feedback to Support Analytics. In *2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–12.
[60] Dong Wang, Yuki Ueda, Raula Gaikovina Kula, Takashi Ishio, and Kenichi Matsumoto. 2021. Can we benchmark Code Review studies? A systematic mapping study of methodology, dataset, and metric. *Journal of Systems and Software* (2021).
[61] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 8696–8708.
[62] Lanxin Yang, Jinwei Xu, Yifan Zhang, He Zhang, and Alberto Bacchelli. 2023. EvaCRC: Evaluating Code Review Comments. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 275–287.
[63] Zezhou Yang, Cuiyun Gao, Zhaoqiang Guo, Zhenhao Li, Kui Liu, Xin Xia, and Yuming Zhou. 2024. A Survey on Modern Code Review: Progresses, Challenges and Opportunities. *arXiv preprint arXiv:2405.18216* (2024).
[64] Yongda Yu, Guoping Rong, Haifeng Shen, He Zhang, Dong Shao, Min Wang, Zhao Wei, Yong Xu, and Juhong Wang. 2024. Fine-tuning large language models to improve accuracy and comprehensibility of automated code review. *ACM transactions on software engineering and methodology* 34, 1 (2024), 1–26.
[65] Farida El Zanaty, Toshiki Hirao, Shane McIntosh, Akinori Ihara, and Kenichi Matsumoto. 2018. An empirical study of design discussions in code review. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (Oulu, Finland) *(ESEM '18)*. Association for Computing Machinery, New York, NY, USA, Article 11, 10 pages.
[66] Qunhong Zeng, Yuxia Zhang, Zeyu Sun, Yujie Guo, and Hui Liu. 2024. COLARE: Commit Classification via Fine-grained Context-aware Representation of Code Changes. In *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 752–763.
[67] Junwei Zhang, Zhongxin Liu, Xing Hu, Xin Xia, and Shanping Li. 2023. Vulnerability detection by learning from syntax-based execution paths of code. *IEEE Transactions on Software Engineering* (2023), 4196–4212.
[68] Zelin Zhao, Zhaogui Xu, Jialong Zhu, Peng Di, Yuan Yao, and Xiaoxing Ma. 2023. The Right Prompts for the Job: Repair Code-Review Defects with Large Language Model. *arXiv preprint arXiv:2312.17485* (2023).
[69] Jieming Zhu, Pinjia He, Qiang Fu, Hongyu Zhang, Michael R. Lyu, and Dongmei Zhang. 2015. Learning to Log: Helping Developers Make Informed Logging Decisions. In *Proceedings of the 37th International Conference on Software Engineering (ICSE '15)*. 415–425.

Received 2025-02-26; accepted 2025-03-31