

# Exploring the Advances in Identifying Useful Code Review Comments

Sharif Ahmed  
Computer Science Department  
Boise State University  
Boise, ID, USA  
sharifahmed@u.boisestate.edu

Nasir U. Eisty  
Computer Science Department  
Boise State University  
Boise, ID, USA  
nasireisty@boisestate.edu

**Abstract**—Effective peer code review in collaborative software development necessitates useful reviewer comments and supportive automated tools. *Code review comments* are a central component of the Modern Code Review process in the industry and open-source development. Therefore, it is important to ensure these comments serve their purposes. This paper reflects the evolution of research on the usefulness of *code review comments*. It examines papers that define the usefulness of code review comments, mine and annotate datasets, study developers' perceptions, analyze factors from different aspects, and use machine learning classifiers to automatically predict the usefulness of *code review comments*. Finally, it discusses the open problems and challenges in recognizing useful *code review comments* for future research.

**Index Terms**—Modern Code Review, Useful Comments, Software Quality, Software Engineering

## I. INTRODUCTION

Software engineers have been using code review for decades to significantly improve code quality, enhance collaboration, facilitate knowledge transfer, enforce coding standards, and save time and cost. Initially, software developers and testers identified software errors and defects in person through a process called *Formal Code Review* or *Fagan Inspection* [1]. This process involved six phases: planning, overview, preparation, inspection meeting, rework, and follow-up. However, with the advancement of the Globalization, software development teams have become more distributed globally, resulting in a decline in the use of this in-person source-code quality inspection process. In addition, the appearance of online communication platforms facilitated developers to conduct such discussions asynchronously and informally. This online tool-based source code inspection technique is known as *Modern Code Review (MCR)*.

The MCR process has been adopted by both industry and open-source developers. However, organizations may customize the process to fit their specific needs, using different tools like GitHub or Gerrit and implementing various policies and working cultures. Generally, the MCR process involves five phases: review request, reviewer se-

lection, review task selection, code review/checking, and review feedback. Reviewers can provide feedback using two approaches. The first is through an annotation on IDE, which is similar to reviewing a paper document with a pen or pencil, as seen in the Rich Code Annotation tool [2]. The second approach is through written feedback in natural language, emoji, emoticons, animation, and votes, which is known as *code review comment*.

The practice of code review aims to produce quality software while saving time and money. However, the intermediate outcome of the code review process, *code review comment*, has the most significant impact on regulating the outcomes. The importance of *code review comments* was demonstrated in Kononenko et al.'s empirical study of the Mozilla code review process [3]. However, Bosu et al. [4] found that 34.5% of the *code review comments* were not useful at Microsoft. Previous studies have defined the usefulness of *code review comments* [4]–[6], mined and annotated datasets of **useful** *code review comments* [6]–[8], studied developers' perceptions in commercial [4] and open-source projects [8], analyzed factors from various aspects [4]–[10], adapted fine-grained taxonomy [8] for useful *code review comments*, and used machine learning classifiers to automatically predict the usefulness of *code review comments* [4]–[7], [10]. Although these studies have progressed the research problem since 2014 (Fig. 1), some works did not reference their previous related papers. This observation has prompted us to reflect retrospectively on the literature on the usefulness of *code review comments*.

To reflect on these studies, we conducted a literature review, focusing on studies explicitly or implicitly exploring the usefulness of *code review comments*. Next, we delved deeper into the studies and evaluated their contributions, approaches, limitations, and takeaways in Sec-II. Our reflection paper is intended to assist researchers and developers in advancing ongoing research or identifying an appropriate strategy for analyzing or identifying the usefulness of *code review comments*.

## II. REFLECTION

In this section, we reflect on previous research on predicting and analyzing the usefulness of *code review*

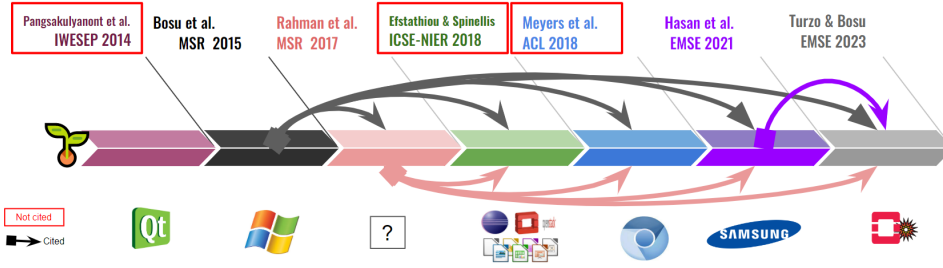


Fig. 1. Progression of Works on Usefulness of *code review comments*

*comments*. We selected a set of studies and refer to these papers as ‘key papers’ (Fig. 1).

**1. Key Paper Selection:** We utilized a multi-step approach to identify the key papers for our study. Initially, we searched Google Scholar using the query string “useful code review comments.” Subsequently, we performed backward and forward snowballing to expand our search further. Regarding inclusion criteria, we considered papers published in peer-reviewed workshops, conferences, or journals. We excluded pre-print papers and magazine articles from our selection process. While many papers explore the efficiency and value of the Modern Code Review (MCR) process, including its various phases, techniques, recommendations, and generation, our focus was explicitly on papers that addressed or analyzed the usefulness of English code review comments. Consequently, we excluded papers that mentioned code review comments without explicitly delving into assessing or identifying their usefulness. As a result of our rigorous selection process, we identified seven studies that met our criteria, and we presented a synthesis of the extracted data in Table I.

**2. Exploration:** Here, we explore different aspects ( $A-J$ ) of code review comments’ usefulness from usefulness definition to automatic classification and its evaluation.

Code Review Comment				
Functional	Refactoring	Documentation	Discussion	False Positive
Defects	Solution Approach		Design	
Logical	Organization		Praise	
Validaton	Alternate Output		Quesiton	
Resource	Identifier Naming			
Timing	Visual Representation			
Support				
Interface				

Fig. 2. Taxonomy of *Code review comments* by Turzo & Bosu [8]

### A. Usefulness Definition

In 2014, Pangsakulyanont et al. [5] established a definition for *code review comments* categorizing them as either **useful** or **useless** according to the degree of similarity between the commit message of the author and the reviewer’s *code review comment* for a given code change. In

cases where the usefulness of the *code review comment* was unclear, they classified it as **undetermined**.

In the following year, Bosu et al. [4] conducted a study at Microsoft, in which seven developers classified *code review comments* as **useful**, **somewhat useful**, and **not useful**. While building a usefulness prediction model, they treated **somewhat useful** as **useful**. The authors defined a *code review comment* as **useful** if it led to a code change in nearby source-code lines. Their experiment investigated the proximity between the code-change and the *code review comment*, with nearness ranging from 1-10 lines. They discovered that changes one line away had the lowest rates of *false positives* and *false negatives* for usefulness classification.

Recently, in 2023, Turzo & Bosu [8] have defined and annotated comments as **useful** if the author explicitly acknowledges the reviewer’s identified issue as good, implicitly acknowledges the feedback by implementing recommended changes, or explicitly retains the feedback for future modifications.

### B. Taxonomy of Code Review Comments & Usefulness









Turzo & Bosu [8] expanded on the labeling of their dataset of *code review comments* beyond the binary **useful** and **not-useful** labels. They introduced 18 categories to gain a deeper understanding of the usefulness of *code review comments*. These categories were adapted from Beller et al.’s [11] taxonomy of *code review comments* for changes. We have redrawn their taxonomy in Fig. 2.

### C. Understanding Developers’ Perception


Researchers have investigated how developers perceive the usefulness of *code review comments* in both commercial [4] and open-source [8] communities. We present a concise overview of their findings below.

a) **Microsoft Study (2015):** Bosu et al. [4] conducted a three-stage experiment at Microsoft to characterize the usefulness of *code review comments*. Firstly, they interviewed developers to gain insight into useful *code review comments*. Secondly, they annotated the usefulness based on developers’ understanding and built a usefulness predictor. Thirdly, they conducted an empirical analysis of the identified factors. Based on prior studies, the authors focused on two types of factors: (i) the *reviewers* and (ii)

TABLE I  
SYNTHESIS OF EXISTING WORKS REGARDING USEFULNESS OF CODE REVIEW COMMENTS

 Papers	Data Source	Input Factors	Contribution	Statistical Analysis	Model/ Classifiers	Validation	Evaluation Metric
[5] Pangsakulyanont et al., 2014	Qt project	 Commit message	Defined Usefulness		Vector-Space-Model	300x10-fold	Accuracy, Precision, Recall
[4] Bosu et al., 2015	Microsoft Azure, Bing, Visual Studio, Exchange, Office	 Textual, Patch, Activity	Defined Usefulness, Perspective, Characterization, Prediction		Decision Tree	100x10-fold	Accuracy, Precision, Recall
[7] Rahman et al., 2017	X- Company	 Textual, Patch, Experience	Feature Enhancement, Prediction, Dataset	Mann Whitney Wilcoxon test, Cohen's D	Naive Bayes, Logistic Regression, Random Forest	1 x 10-fold	Accuracy, Precision, Recall F1
[9] Efstathiou & Spinellis, 2018	Eclipse, Libre-Office, AOSP, OpenStack	 Textual, Patch	Preliminary work on Linguistics		n/a	n/a	Distribution
[6] Meyers et al., 2018	Chromium Conversations Browser & OS	 Linguistic	Linguistic feature exploration, Prediction, Dataset,	Kappa, Recursive Feature Elimination with Cross Validation (RFECV)	Logistic Regression	10x10-fold	Precision, Recall, F1, AUC
[10] Hasan et al., 2021	Samsung R&D Bangladesh	 Textual, Patch, Activity, Experience	Feature Enhancement, Prediction	Pearson Correlation, RFECV	Decision Tree, Logistic Regression, Support Vector Machine, XGBoost, Random Forest, Multilayer Perceptron	20x10-fold	Accuracy, Precision, Recall, F1
[8] Turzo & Bosu, 2023	OpenDev Nova	 Activity, Experience	Characterization, Perspective, Dataset, Questionnaire Revamped Taxonomy	Kappa, Chi-Square, Sarle's Variable Clustering (VURCLUS)	Linear Regression, Multinomial Logistic Regression	n/a	R <sup>2</sup>

the *changeset*. The study suggests that expert reviewer selection is crucial, but the inclusion of novice reviewers can also be beneficial for disseminating expertise. The results also indicate that review effectiveness decreases as the number of files in the patches increases.

b)  *OpenDev Study (2023)*: Turzo & Bosu [8] conducted a three-stage experiment to gain a deeper understanding of the usefulness of *code review comments*. Firstly, they manually categorized and labeled the usefulness of *code review comments*. Secondly, they obtained insight from OSS developers through an online survey. Thirdly, they performed a regression analysis on contextual and participant factors with the usefulness of *code review comments*. Kononenko et al. [12] performed a similar analysis for buggy changes at *Mozilla*. Turzo & Bosu obtained 160 usable empirical responses from OpenDev developers via email while maintaining the IRB protocol. The OpenDev developers suggested that usefulness depends on verbal and process aspects in addition to technical contributions. They also emphasized that functional *code review comments* matter the most and hold diverse views on the usefulness, including praise, documentation, design discussion, resource synchronization, and visual representation.

#### D. Factors and Features

Initially, Pangsakulyanont et al. [5] employed the Vector Space Model with cosine similarity measure between the commit message and the *code review comment* on a code-change to classify usefulness.

Subsequently, in 2015, Bosu et al. [4] identified the factors of useful *code review comments* through an empirical study at Microsoft and defined “usefulness”. They also developed a Decision Tree Classifier to model a usefulness classifier. To generate features for the classifier, they utilized textual properties of *code review comments* and attributes of review activities (Fig. 3).

In another study, Kononenko et al. [12] conducted an empirical investigation on the open-source project *Mozilla*. They discovered that 54% of the reviewed changes failed to detect bugs in the code. They also introduced factors, such as code author and reviewer experience, that impact the quality of code reviews. Rahman et al. [7] collected *code review comments* from commercial projects and developed a model to predict their usefulness. Their approach involved using textual properties of the *code review comments* and features related to the developers’ experience (Fig. 3). However, unlike Bosu et al. [4], they did not incorporate any features related to the code review activities.

Previous studies, such as [4], [7], [10], have achieved better results by including textual features and developers’ experience and review activity features. However, in projects with no prior review or record of reviewers and developers, these important features will be missing in these studies. Efstathiou and Spinellis [9] proposed measuring usefulness using linguistic semantics to address this issue. Concurrently, Meyers et al. [6] used several linguistic features, including “text complexity,” “density,” “formality,” “politeness,” “sentiment,” and “uncertainty,” to classify *code review comments*. Recently, OSS developers

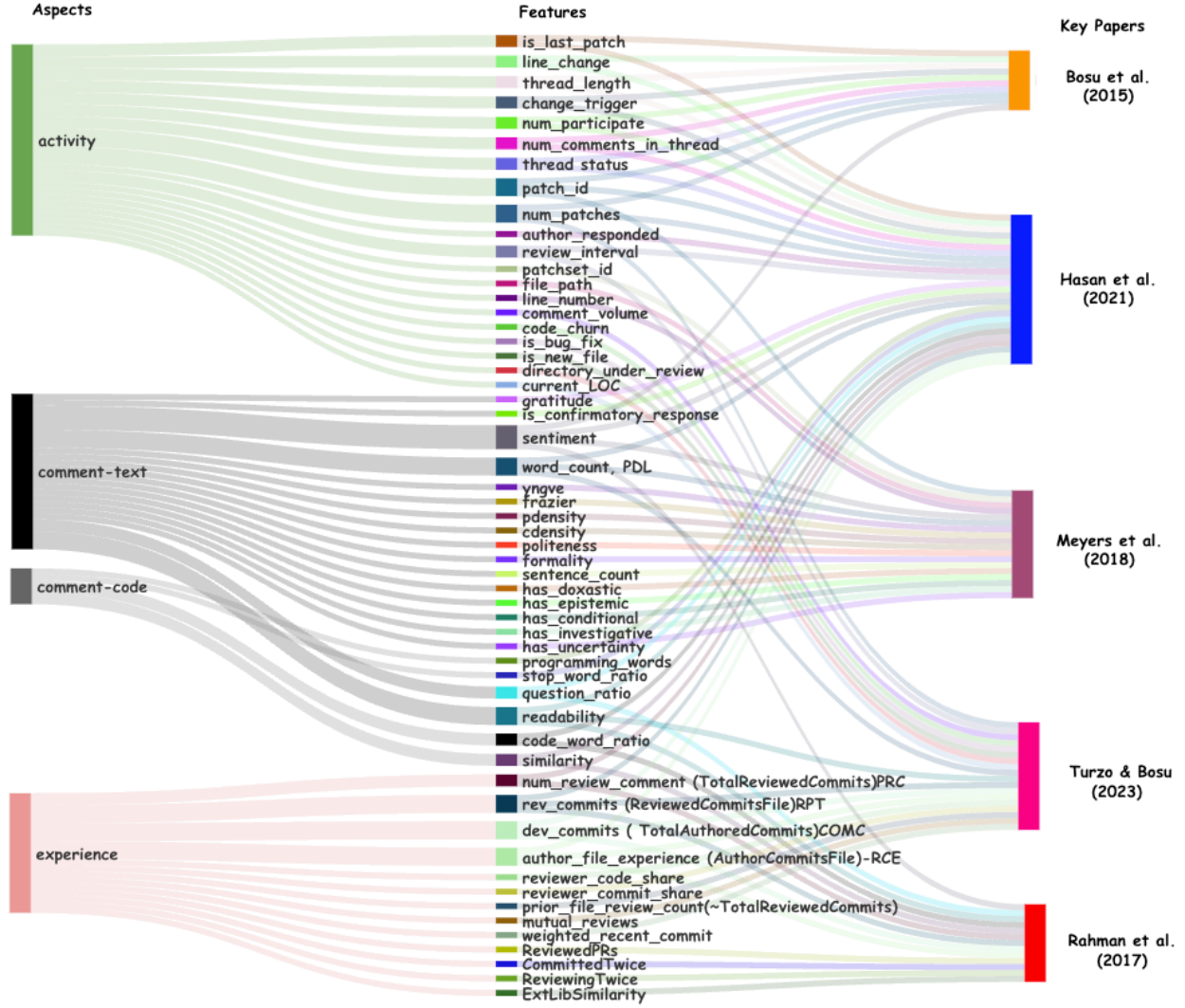


Fig. 3. An examination of the features employed to date for predicting/ analyzing the usefulness of *Code Review comments*.

also stated politeness and comprehension as key factors [8].

Hasan et al. [10] developed an in-house web-based application to reward reviewers at *Samsung Research Bangladesh* for their useful feedback. They did not consider any linguistic features from Meyers et al. [6]. The authors used features from three aspects of prior works: textual, experience, and review activities to predict the usefulness of reviewers' *code review comments*. Additionally, they extended the existing features by incorporating features related to code review activities (Fig. 3). They also introduced a new textual feature called word count to identify short comments. Furthermore, they presented several features for predicting the usefulness of *code review comments* from the literature and their work, but these features are limited to their model.

We have compiled all features and categorized them based on their aspects, such as review context or activity, developers' experience, and textual features from

*code review comments*. Additionally, we have divided the comment-textual features into *comment-text* and *comment-code* aspects. The features mapped from various aspects to key papers are presented in Fig. 3. For instance, Fig. 3 shows that the 'change\_trigger' feature used by Bosu et al. [4] and Hasan et al. [10] was derived from the code review 'activity' aspect.

#### E. Available Data Sources

The dataset by Bosu et al. [4] and Hasan et al. [10] cannot be made public due to the Non-Disclosure Agreements (NDAs) they have with Microsoft and Samsung R&D Bangladesh, respectively. However, we have identified three other datasets relevant to the usefulness of *code review comments*. The statistics of usefulness labels for all available datasets are presented in Fig. 4.

1) *RevHelper*: In 2017, Rahman et al. [7] used Github API to collect *code review comments* from four commercial subject systems of an anonymous company. They manually



annotated each *code review comment* as either **useful** or **non-useful**, based on the explanation of usefulness provided in the prior work by Bosu et al. [4]. The dataset includes 879 **useful** and 602 **not-useful** comments. The authors distributed the dataset into two sets: a *train-test set* and a *validation set*.

2) *Chromium Conversation*: In 2018, Meyers et al. [6] developed a dataset from code review in Google Chromium Project. The open-source code review tool used in Chromium is known as *Rietveld*. Using RESTful API, the authors obtained 2,855,018 publicly accessible code review comments from 2008 to 2016. They then selected comments posted by reviewers. Next, they automatically identified acted-upon *code review comments* by exploiting Rietveld’s “Done” click feature. Their annotation principle is similar to the RevHelper [7]. The dataset consists of 2994 **acted-upon** (i.e., **useful**) and 800 **not-(known to be)-acted upon** *code review comments*.

3) *OpenDev Comment*: In 2023, Turzo & Bosu [8] developed a dataset from OpenDev’s Nova OSS project. They utilized the Gerrit Miner tool to mine 795,226 publicly available code-review records from 2011 to 2022. The resulting dataset consists of 2,052 **useful** and 602 **not-useful** *code review comments* that were manually annotated from 18 categories and 5 comment groups.

*Annotation Verification*: The authors of the dataset papers also verified their annotations differently. Rahman et al. manually annotated the RevHelper [7] dataset and conducted a random cross-check. Meyers et al. [6] automatically annotated **acted-upon** *code review comments* and manually verified 23% of the sample size, resulting in an inter-annotator agreement score of 0.89. They then manually inspected the 2047 **not acted-upon** *code review comments* that were automatically considered and included 800 correctly identified **not acted-upon** *code review comments* in their final dataset. Turzo and Bosu [8] reported inter-rater reliability scores of 0.68 and 0.84 for their manual *comment-category* and *comment-usefulness* annotation, respectively.

#### F. Feature Analysis and Selection

Rahman et al. [7] compared the effectiveness of input features with that of *code review comments* in predicting the usefulness of the latter. They utilized the *Mann Whitney Wilcoxon test* to identify differences in features between useful and non-useful *code review comments*, and *Cohen’s D* as an effect-size measure. In contrast, Hasan et al. [10] used the *Pearson Correlation* measure to interpret their input features. In their study, Rahman et al. [7] compared the textual characteristics of useful and non-useful comments. They found that the *code element ratio*, *stop word ratio*, and *conceptual similarity* were statistically significant, while *reading ease* and *question ratio* were not. Additionally, they explored the relationship between developers’ experience and useful *code review comments*. They found that *code ownership* and *reviewership* were

statistically significant, while the relationship between developers’ experience and usefulness was not straightforward, which is consistent with Bosu et al. [4].

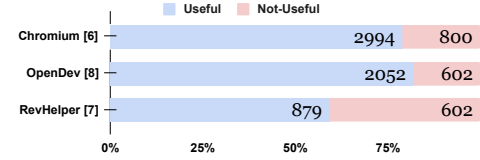


Fig. 4. Available Datasets [6]–[8]

In 2018, Efstathiou and Spinellis [9] conducted a preliminary examination using the textual data of review messages obtained from four open-source projects (Table I). Their analysis revealed a possible avenue for exploring linguistic semantics to identify beneficial *code review comments*; however, verifying the applicability would require the input of skilled linguists.

Meyers et al. [6] used Recursive Feature Elimination with Cross-Validation (RFECV) to filter out the less contributing features. Hasan et al. [10] also used RFECV after excluding highly correlated features.

Turzo & Bosu [8] initially applied Sarle’s Variable Clustering (VURCLUS) technique to identify highly correlated features. They then used Spearman’s correlational hierarchical clustering approach to remove features with a high correlation coefficient of 0.7+ from their regression model. They also used the Chi-Square test to check the relationship between respondents’ demographics and responses.

#### G. Models

Three key papers employed only a single algorithm to classify useful *code review comments*. Specifically, Pangsakulyanont et al. [5] used the Vector Space Model to measure similarity, Bosu et al. [4] employed the Decision Tree Classifier, and Meyers et al. [6] utilized the Logistic Regression Classifier. Other studies explored multiple classification algorithms, such as the Decision Tree Classifier [4], [10], Naive Bayes Classifier [7], Logistic Regression Classifier [6], [7], [10], Random Forest Classifier [7], [10], Support Vector Machine [10], Multilayer Perceptron Classifier [10], and eXtreme Gradient Boosting Classifier [10]. Two studies [7], [10] found Random Forest Classifier as the best performing model in their evaluation. For regression analysis, Turzo & Bosu [8] used Linear Regression and Multinomial Logistic Regression models.

All previous models had imbalanced training datasets except for Hasan et al. [10] who employed the Synthetic Minority Over-sampling Technique (SMOTE).

#### H. Evaluation Metrics

TABLE I indicates that the majority of key papers evaluated the effectiveness of their code review comment usefulness prediction using the **accuracy**, **precision**, **recall**, and **F1-measure** metrics. One of the papers also reported using the **AUC** (Area Under the Curve) metric

as an evaluation method. Another study utilized  $R^2$  to measure the efficacy of their regression models.

Though they used different datasets and evaluation measures, the **accuracy** and  $F1_{w.avg.}$  found in the key papers range from ~63 to ~87%. We also observed that they could achieve ~6 to ~10% higher accuracy than the majority class (i.e., % useful comments).

### I. Model Comparison

Rahman et al. [7] conducted a comparison with several simulated variants of their previous work [4]. Similarly, Hasan et al. [10] replicated two of their earlier models [4], [7] with slight modifications and compared their performance. Their replication addressed the issue of inaccessible features such as *external library*, *keyword ratio*, *reading ease* in RevHelper [7]. However, they also obtained some features by replacing new tools with the original ones. For example, while Bosu et al. [4] used a text-sentiment analyzer, Hasan et al. [10] replaced it with a code-review sentiment analyzer in their replication.

Various techniques, including cross-validation, offer a way to mitigate the risk of over-fitting or selection bias in model development. These techniques also provide valuable insights into the generalizability of a model, particularly its ability to perform accurately on an independent dataset. All of the prediction works [5], [4], [7], [6], and [10] evaluated their models using 10-fold cross-validation, with 300, 100, 0, 10, and 20 repetitions, respectively.

### J. Limitations

The literature exhibits two major problems.

1) *Data Unavailability or Limited Availability*: Bosu et al. [4] and Hasan et al. [10] could not share their dataset for proprietary restrictions. Due to the same reason, Rahman et al. [7] shared a partial dataset, and the code changes were excluded. However, the other two accessible [6], [8] datasets have identifiers instead of direct code changes. Unfortunately, due to time constraints, it was not feasible to assess the viability of retrieving the code changes from the given linking identifiers. However, the existing usefulness prediction models [4], [7], [10] demand code-changes (aka patches) to classify usefulness which is absent in available datasets.

2) *Review Activity and Experience Records*: The existing usefulness prediction models [4], [7], [10] require review activity, developer experiences, or both, which may not be available for a new developer, project, or company.

## III. DISCUSSION

Researchers have attempted to define the usefulness of code review comments, study developers' perceptions, analyze underlying factors, extract features from these factors, build datasets, and classify the usefulness of code review comments. However, due to proprietary restrictions, most studies have not made their experimental datasets available [4], [7], [10]. These unavailable data limit the replication process and public reviews.

While examining existing features, we considered *comment-code* as a new aspect. New potential features can be adopted under this aspect to predict the usefulness of *code review comments*. Crucially, a key paper suggested the linguistic approach to identify the usefulness of *code review comments* [9]. Cutting-edge NLP tools can be applied accordingly. Furthermore, researchers have evaluated the performance of their models using various metrics. Based on the reported performance, it is open that there is still room for improvement in the existing models.

Some works were not mentioned in their subsequent papers (Fig. 1), and several reasons could exist. One possible explanation is that two papers were published in the same year. Another reason could be using different category names for usefulness [6] or publication in another venue.

Two empirical studies have corroborated many of the findings from other studies, both empirical and non-empirical. These findings include the importance of factors such as reviewer experience, text-readability, timeliness, sentiment, and reviewers' tenure, as well as the inverse relationship between the usefulness of *code review comments* and the size of the changes being reviewed. MacLeod et al. [13] reported challenges and best practices for the MCR process. They found factors of *code review comments* such as *comprehension*, *being respectful* affect the MCR process, and these are relatable to the features adopted in the key papers [4], [7] such as readability, sentiment, etc.

The findings regarding developer experience and its relationship with the usefulness of *code review comments* have been more mixed, with the Microsoft study finding a positive association between reviewers' tenure and usefulness. In contrast, the OpenDev study found a negative association with project tenure. To better understand these discrepancies or discover new factors, it may be useful for researchers to conduct qualitative studies with developers and to perform further data mining analyses.

## IV. CONCLUSION

This paper reflects existing literature and outlines the research trajectory on the usefulness of *code review comments*. It identifies ten areas, including defining the usefulness of *code review comments*, adapting taxonomies, exploring developer perceptions, extracting factors and features, data mining and annotation, conducting feature analysis, predicting usefulness, evaluating approaches, and major limitations. Moving forward, developers may consider creating a tool that can identify not-useful *code review comments* in advance to prevent before commenting. Other related tasks, such as automatic code review comment generation, reviewer or developer recommendation, static analysis, etc., involve *code review comments*. These tasks can leverage the useful *code review comments* prediction models and enhance the effectiveness of the respective tasks or the entire code review process.

## REFERENCES

- [1] M. Fagan, “Design and code inspections to reduce errors in program development,” in *Software pioneers*. Springer, 2002, pp. 575–607.
- [2] R. Priest and B. Plimmer, “Rca: experiences with an ide annotation tool,” in *Proceedings of the 7th ACM SIGCHI New Zealand chapter’s international conference on Computer-human interaction: design centered HCI*, 2006, pp. 53–60.
- [3] O. Kononenko, O. Baysal, and M. W. Godfrey, “Code review quality: How developers see it,” in *Proceedings of the 38th international conference on software engineering*, 2016, pp. 1028–1038.
- [4] A. Bosu, M. Greiler, and C. Bird, “Characteristics of useful code reviews: An empirical study at microsoft,” in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 146–156.
- [5] T. Pangsakulyanont, P. Thongtanunam, D. Port, and H. Iida, “Assessing mcr discussion usefulness using semantic similarity,” in *2014 6th International Workshop on Empirical Software Engineering in Practice*. IEEE, 2014, pp. 49–54.
- [6] B. S. Meyers, N. Munaiah, E. Prud’hommeaux, A. Meneely, J. Wolff, C. O. Alm, and P. Murukannaiah, “A dataset for identifying actionable feedback in collaborative software development,” in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2018, pp. 126–131.
- [7] M. M. Rahman, C. K. Roy, and R. G. Kula, “Predicting usefulness of code review comments using textual features and developer experience,” in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 215–226.
- [8] A. K. Turzo and A. Bosu, “What makes a code review useful to opendev developers? an empirical investigation,” *Empirical Software Engineering*, 2023.
- [9] V. Efstathiou and D. Spinellis, “Code review comments: language matters,” in *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, 2018, pp. 69–72.
- [10] M. Hasan, A. Iqbal, M. R. U. Islam, A. Rahman, and A. Bosu, “Using a balanced scorecard to identify opportunities to improve code review effectiveness: an industrial experience report,” *Empirical Software Engineering*, vol. 26, no. 6, pp. 1–34, 2021.
- [11] M. Beller, A. Bacchelli, A. Zaidman, and E. Juergens, “Modern code reviews in open-source projects: Which problems do they fix?” in *Proceedings of the 11th working conference on mining software repositories*, 2014, pp. 202–211.
- [12] O. Kononenko, O. Baysal, L. Guerrouj, Y. Cao, and M. W. Godfrey, “Investigating code review quality: Do people and participation matter?” in *2015 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 2015, pp. 111–120.
- [13] L. MacLeod, M. Greiler, M.-A. Storey, C. Bird, and J. Czerwinka, “Code reviewing in the trenches: Challenges and best practices,” *IEEE Software*, vol. 35, no. 4, pp. 34–42, 2017.