Science of Computer Programming 208 (2021) 102652

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

A review of code reviewer recommendation studies: Challenges and future directions

H. Alperen Çetin *, Emre Doğan, Eray Tüzün

Department of Computer Engineering, Bilkent University, Ankara, Turkey

ARTICLE INFO

Article history:
Received 2 October 2020
Received in revised form 29 March 2021
Accepted 29 March 2021
Available online 14 April 2021

Keywords:
Systematic literature review
Code reviewer recommendation
Reviewer recommendation
Modern code review
Pull request

ABSTRACT

Code review is the process of inspecting code changes by a developer who is not involved in the development of the changeset. One of the initial and important steps of code review process is selecting code reviewer(s) for a given code change. To maximize the benefits of the code review process, the appropriate selection of the reviewer is essential. Code reviewer recommendation has been an active research area over the last few years, and many recommendation models have been proposed in the literature.

In this study, we conduct a systematic literature review by inspecting 29 primary studies published from 2009 to 2020. Based on the outcomes of our review: (1) most preferred approaches are heuristic approaches closely followed by machine learning approaches, (2) the majority of the studies use open source projects to evaluate their models, (3) the majority of the studies prefer incremental training set validation techniques, (4) most studies suffer from reproducibility problems, (5) model generalizability and dataset integrity are the most common validity threats for the models and (6) refining models and conducting additional experiments are the most common future work discussions in the studies.

1. Introduction

Code review is the inspection of code changes by developers other than the committer of the change. When performed appropriately, code review increases software quality and helps early detection of software defects [1]. Over the last decades, the code review process has evolved from code inspection meetings [2] into a tool-based code review process, also known as *modern code review* [3].

Modern code review is one of the most frequently applied practices in software development [4]. In addition to its primary motivations such as improving the code quality and finding defects, it is also helpful to transfer knowledge within the development team, increase the team awareness and share code ownership [5,6]. Leading software companies apply code review practice in their development life cycle and share the challenges faced during the code review process and the best practices [3,5].

One of the important steps of code review is the selection of the correct reviewers for code changes. In a typical software project, code reviewers are assigned to incoming changes (e.g., pull requests) manually by team leaders or integrators. The manual nature of this task might pose two potential problems. First, since this is a manual task, this may lead to latency

* Corresponding author.
E-mail addresses: alperen.cetin@bilkent.edu.tr (H.A. Çetin), emre.dogan@bilkent.edu.tr (E. Doğan), eraytuzun@cs.bilkent.edu.tr (E. Tüzün).

https://doi.org/10.1016/j.scico.2021.102652
0167-6423/

Table 1
Steps of our SLR process [12].

Step 1	Identification of the need for a review	**Planning the Review**
Step 2	Commissioning a review	
Step 3	Specifying the research questions	
Step 4	Developing a review protocol	
Step 5	Evaluating the review protocol	
Step 6	Identification of research	**Conducting the Review**
Step 7	Selection of primary research	
Step 8	Study quality assessment	
Step 9	Data extraction and monitoring	
Step 10	Data synthesis	
Step 11	Specifying dissemination mechanisms	**Reporting the Review**
Step 12	Formatting the main report	
Step 13	Evaluating the report	

in processing incoming code changes. Secondly, the manual selection might not end up with the appropriate code reviewer selection for an incoming code change, which might lead to the rejected review requests or inefficient code reviews [5].

According to the findings of Thongtanunam et al. [7], code review processes with an inappropriate reviewer assignment take longer to complete. To avoid such cases, several code reviewer recommendation (CRR) models have been proposed by researchers from both academia and industry. The common goal of these studies is to automatically find and recommend the most suitable reviewer for a given code changeset.

In the last six years, a significant number of CRR studies (more than four studies per year on average and seven studies in 2020) have been published in a wide range of journals and conferences, and recently CRR research draws the interest of the software engineering research community. On the industry side, the recent usage of automated reviewer recommendation algorithms in code review tools such as Gerrit [8], Upsource [9], Crucible [10] and Github [11] also indicates widespread adoption in practice.

The majority of the studies propose disparate approaches utilizing different categories of algorithms using combinations of heuristic and machine learning methods. Some of the studies have significant problems such as lack of reproducibility. To date, a sufficient number of studies have been published to facilitate identifying relations among them. To properly summarize the current state of the art in CRR studies and indicate the points that can help future CRR researchers and industry practitioners, we perform a systematic literature review (SLR) of the 29 CRR studies published from 2009 to 2020.

The major contributions of our study are:

- Presenting the different methods/algorithms used.
- Categorizing the datasets and artifacts used.
- Showing the different evaluation metrics and validation setups used.
- Going through an evaluation of their reproducibility.
- Summarizing the threats to validity in CRR studies.
- Suggesting future research directions for CRR studies.

The remainder of this paper is organized as follows. The next section describes our SLR methodology and research questions. Section 3 presents our findings for answering the research questions. Section 4 provides an in-depth discussion by summarizing the implications for the research community & industry, reproducibility & data integrity issues and prevalent differences in CRR studies. Section 5 discusses the threats to validity for our findings. Section 6 gives brief information about related work. The final section concludes the paper with a summary of the contributions.

2. Methodology

The steps followed in this SLR are shown in Table 1. Due to the growing interest of CRR in the academia and the industry (see Section 1), and considering the fact that there is no other secondary study investigating the state of CRR studies (see Section 6), the need for a systematic review of CRR studies arises to summarize the current state of the art in the CRR studies (*Step 1*). After this need was identified, we came up with a list of research questions (RQs) as given in Table 2 (*Step 2-3*). Then, a proper review protocol was structured, as presented in Section 2.1 (*Step 4-5*). Section 2.2 gives the details of our search strategy (*Step 6*). Inclusion and exclusion criteria are discussed in Section 2.3 (*Step 7*).

Our main criteria for quality assessment was whether the studies were providing empirical evidence (*Step 8*). Section 2.4 reveals the details of the data extraction process (*Step 9*). Finally, the synthesis of our findings is shared in Section 3 (*Step 10*).

Table 2
Research questions and their motivations.

	Research questions	Motivations
RQ1	What kind of methods/algorithms are used in CRR studies?	Identifying the different high-level categories of CRR methods. Investigating the recent trends in terms of these categories.
RQ2	What are the characteristics of the datasets used in CRR studies?	Identifying dataset types and data sources. Identifying the diversity of the artifacts used in CRR models.
RQ3	What are the characteristics of the evaluation setups used in CRR studies?	Identifying the different validation techniques. Identifying the evaluation metrics.
RQ4	Are the models proposed in CRR studies reproducible?	Investigating whether the datasets are available or not. Investigating whether the details of the algorithms are shared or not.
RQ5	What kind of threats to validity are discussed in CRR studies?	Identifying the validity threats mentioned and discussed in CRR studies. Providing an overview of the common threats to validity.
RQ6	What kind of future works are discussed in CRR studies?	Identifying future works mentioned in CRR studies. Providing an overview of the common future works.

2.1. Review method

In order to review the literature on CRR studies, a systematic methodology was applied. Systematic literature review (or systematic review) is a methodology to identify, evaluate and interpret all available research studies regarding a particular research topic [12]. This SLR follows the guidelines of Kitchenham and Charters [12] on SLR methodology in software engineering.

2.2. Search strategy

The search process consisted of activities such as deciding which digital libraries to use, defining the ideal search string and collecting the initial list of primary studies [12]. In order not to miss any significant studies, the most popular literature databases were searched. The searched databases are given below:

- ACM Digital Library[1]
- IEEE Xplore[2]
- ScienceDirect[3]
- Springer[4]
- ISI Web of Knowledge[5]

The search string used in these databases was structured with respect to the following steps:

1. Identification of the search terms from titles, keywords and abstracts of known CRR studies.
2. Identification of the search terms from RQs.
3. Identification of the synonyms and alternatives of the terms.
4. Structuring the final search string using ORs and ANDs.

Since the syntax rules of the digital libraries to create search strings differ, a modified version of the following search string was created and used in order to search each database:

```
((code OR peer OR "pull request" OR patch) AND reviewer AND
(recommender OR recommending OR recommendation OR suggesting))
```

We tried to be consistent with searching the queries on *the titles and abstracts* of the studies. If a digital library did not allow us to search on the title and abstract level, then we searched on the whole text.

[1] dl.acm.org.
[2] ieeexplore.ieee.org.
[3] sciencedirect.com.
[4] springerlink.com.
[5] apps.webofknowledge.com.

Table 3
Inclusion and exclusion criteria defined for study selection process.

Categories	Criteria
Inclusion	Studies that propose a CRR method / algorithm. Journal versions are included for the studies that have both conference and journal versions.
Exclusion	Studies written in a language other than English. Studies without empirical results. Studies with fewer than five pages.

Table 4
Data items extracted from the primary studies.

Data items extracted	Description of data item	Related RQ(s)
Title	The title of the study	Overview
Year	The publication year of the study	Overview
Venue	The venue in which the study was published	Overview
Venue type	Journal, conference, workshop or grey	Overview
Author names	The names of the authors of the study	Overview
Author affiliations	University, industry or both	Overview
Method name	The name of the method as mentioned in the study	RQ1
Method summary	A summary of the method proposed in the study	RQ1
Method type	Machine learning, heuristic or hybrid	RQ1
Dataset type	Open source, closed source or both	RQ2, RQ4
Dataset details	The number of projects and the number of pull requests	RQ2, RQ4
Dataset source	Github, Gerrit, Bugzilla, not mentioned or other	RQ2, RQ4
Artifacts used	Software artifacts used in the studies	RQ2, RQ4
Metrics used	Evaluation metrics used in the studies	RQ3, RQ4
Validation technique	Validation technique used in the experiments	RQ3, RQ4
Future works	Future works mentioned in the study	RQ5
Threats to validity	Threats to validity mentioned in the study	RQ6

After the search process was completed, the references (backward snowballing) and the citations (forward snowballing) of the resulting papers were manually checked in order to include any studies missing from our search. In the snowballing process, Google Scholar[6] was used.

2.3. Study selection

In order to select the relevant primary studies published until November 2020, we defined and applied a set of inclusion and exclusion criteria to the initial list of primary studies. These criteria are shown in Table 3.

To avoid missing any studies, two authors performed separate searches on the selected databases, then merged their results in order to form the final list of primary studies. When there was a disagreement on the selection of the primary studies, the third author was consulted, and the problems were discussed until we came to a consensus.

A detailed description of the search and selection process of primary studies is given in Fig. 1. The numbers within the parentheses next to the digital library names correspond to the number of initial studies resulting from each digital library. The primary studies consist of papers that include a CRR model/algorithm and share some experimental results. We also exclude the studies not in English and shorter than five pages. So, we focus on the full studies that contain a model and a proper evaluation. If the same study had more than one version (e.g., a journal extension of a paper appearing in conference proceedings), the journal version was chosen.

The final list of 29 primary studies can be found in Appendix A.

2.4. Data extraction

The selected primary studies were analyzed to collect data according to the RQs given in Table 2. We created a data extraction form to retrieve the necessary information to answer the RQs. Table 4 shows the data items extracted from the primary studies and RQs related to them. We share the extracted form and the details of each primary study online.[7] In Section 3, the RQs are answered by analyzing the results of the data extraction process.

3. Results

In this section, we share our results for the RQs defined in Table 2. All references in P# notation refer to the primary studies in Appendix A.

[6] https://scholar.google.com/.
[7] https://figshare.com/articles/dataset/Data_Extraction_Form/13439366.

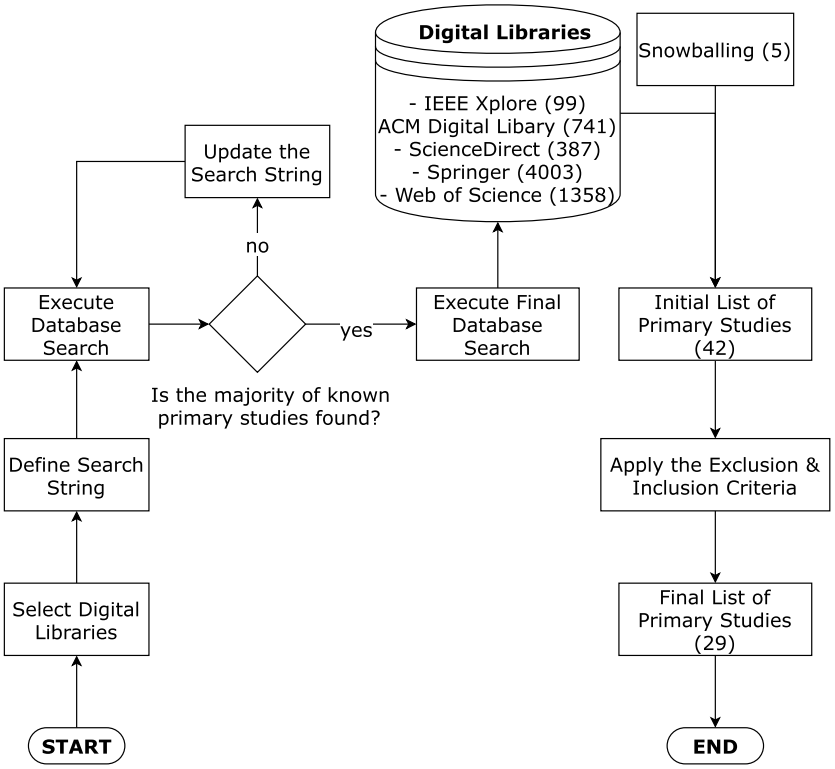


Fig. 1. Detailed process of primary study selection. (The numbers within the parentheses correspond to the number of studies.)

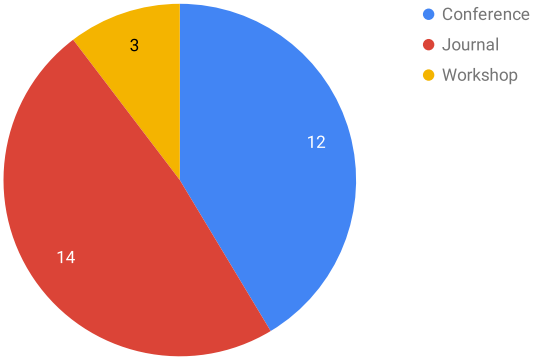


Fig. 2. Venue types. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

As of November 2020, there are a total of 29 CRR primary studies. The first CRR study was published in 2009 [13]. Even though the first one was published 11 years ago, the majority of the studies (26 out of 29) have been published in the last six years. The authors of 23 studies are only affiliated to a university. All authors of two CRR studies are only affiliated to an industrial company. Four studies have authors from both industry and academia.

Fig. 2 shows a pie chart for the venue types of the primary studies. The *journal* category includes the studies published in periodically publishing journals. The *conference* category refers to the studies that are published in conference proceedings, and *workshop* papers are the proceedings for the workshops hosted in conferences. The most common venue is journals with 14 studies followed by 13 conference proceedings and three workshop papers. Note that our methodology affects the numbers since we include journal versions of the studies that have both journal and conference versions.

From Section 3.1 to Section 3.6, we provide our findings to address the RQs defined in Table 2.

3.1. Method/algorithm characteristics (RQ1)

There are many different types of algorithms used in CRR studies. To classify the different types, we define the *machine learning*, *heuristic* and *hybrid* categories.

Table 5
Popular machine learning algorithms used by multiple CRR studies.

Algorithm	Number of studies	Studies
Collaborative filtering	3	P1, P26, P29
Genetic algorithm	3	P4, P25, P27
Support-vector machine	3	P5, P14, P15
Naive Bayes	3	P14, P21, P23
Latent Dirichlet allocation	2	P13, P19
Random forest	2	P14, P23

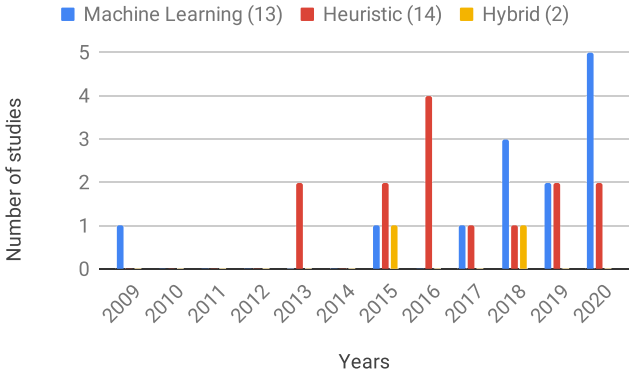


Fig. 3. Methods/algorithms by years. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

The *machine learning* category includes the studies using any data-driven machine learning technique such as support vector machine [P15], genetic algorithm [P4, P25, P27], collaborative filtering [P1, P26, P29], Bayesian network [P9], latent Dirichlet allocation [P13, P19], and Naive Bayes [P21]. For instance, Jeong et al. [P9] extract some features from pull requests and issue artifacts and recommend reviewers by using a Bayesian Network, and Júnior et al. [P14] use different machine learning techniques (Naive Bayes, decision tree, random forest, *k*-nearest neighbor, support vector machine) to recommend reviewers. Table 5 shows the popular machine learning algorithms used by multiple CRR studies.

Heuristic approaches are any kind of problem-solving and practical methods such as graph approaches and mathematical expressions which calculate scores to select the appropriate code reviewers. These heuristic scores are calculated by using graph structures [P8, P11, P18, P20], developer expertise [P2, P6, P22, P24], line change history [P3], text similarity on pull requests [P16, P17] and file path similarity [P7, P12, P16]. For example, Sülün et al. [P20] construct a software traceability graph by using several artifacts, then use a heuristic metric, *know-about*, to calculate familiarity between developers and artifacts.

Methods using both machine learning and heuristic approaches fall under the last category, *Hybrid* approaches. For example, Xia et al. [P10] combine two approaches: text-mining and similarity. They construct a text-mining model by using Multinomial Naive Bayes and a heuristic-based model to find the similarity between the new file path and the previous paths.

Fig. 3 shows the category distribution of algorithms used in CRR studies by years. 14 studies only use heuristic approaches while 13 of them propose machine learning algorithms, and two studies propose hybrid approaches (a mixture of both heuristics and machine learning techniques). Table 6 shows the title, method name and method (algorithm) type for each CRR study.

3.2. Dataset characteristics (RQ2)

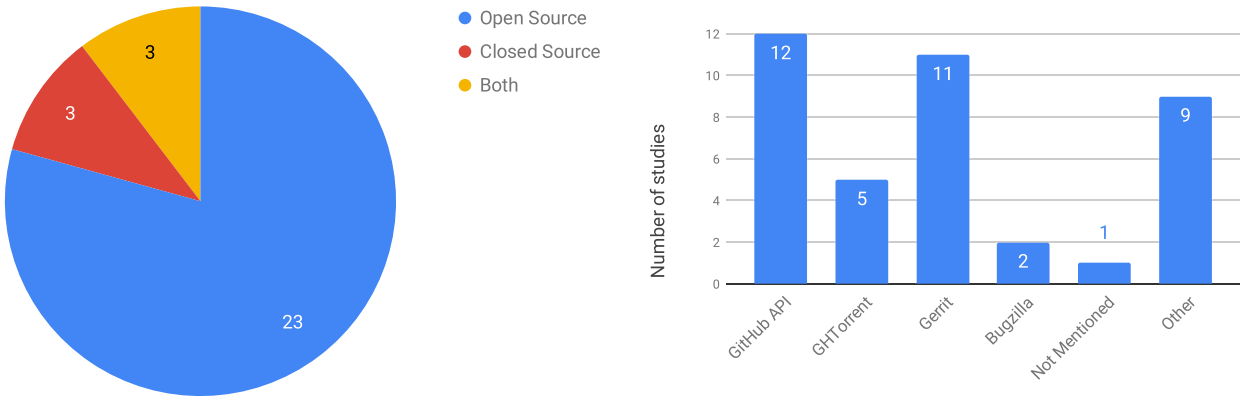
Each study evaluates its proposed model on code review datasets. We first share statistics about the types and sources of these datasets. Then, we investigate what kind of software artifacts are used in CRR studies.

3.2.1. Dataset types and sources

In CRR studies, the datasets consist of two types of projects: *open source* and *closed source*. Open source project datasets are from public repositories and closed source project datasets are from private repositories of closed source projects. Some of the studies use datasets from *both* open and closed source projects. Fig. 4a shows the dataset types used in the studies. Three studies [P3, P24, P26] use closed source datasets and three other studies [P2, P4, P6] use both closed source and open source datasets while the rest of the studies only use open source datasets.

Table 6
Titles, method names and method types of CRR studies (ML: Machine Learning).

Study ID	Study title	Method name	Method type
[P1]	A hybrid approach to code reviewer recommendation with collaborative filtering [14]	PR-CF	ML
[P2]	CoRReCT: code reviewer recommendation in GitHub based on cross-project and technology experience [15]	CORRECT	Heuristic
[P3]	Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation [16]	Review Bot	Heuristic
[P4]	WhoReview: A multi-objective search-based approach for reviewers recommendation in modern code review [17]	WhoReview	ML
[P5]	RevRec: A two-layer reviewer recommendation algorithm in pull-based development model [18]	RevRec	Hybrid
[P6]	Automatically Recommending Peer Reviewers in Modern Code Review [19]	cHRev	Heuristic
[P7]	Who Should Review My Code? A File Location-Based Code-Reviewer Recommendation Approach for Modern Code Review [7]	REVFINDER	Heuristic
[P8]	Patch Reviewer Recommendation in OSS Projects [20]	-	Heuristic
[P9]	Improving Code Review by Predicting Reviewers and Acceptance of Patches [13]	-	ML
[P10]	Who Should Review This Change? Putting Text and File Location Analyses Together for More Accurate Recommendations [21]	TIE	Hybrid
[P11]	Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment? [22]	CN	Heuristic
[P12]	Profile based recommendation of code reviewers [23]	Tversky No Ext	Heuristic
[P13]	Understanding Review Expertise of Developers: A Reviewer Recommendation Approach Based on Latent Dirichlet Allocation [24]	-	ML
[P14]	Automatic Assignment of Integrators to Pull Requests: The Importance of Selecting Appropriate Attributes [25]	-	ML
[P15]	CoreDevRec: Automatic Core Member Recommendation for Contribution Evaluation [26]	CoreDevRec	ML
[P16]	Learning to Rank Reviewers for Pull Requests [27]	-	Heuristic
[P17]	Who Should Comment on This Pull Request? Analyzing Attributes for More Accurate Commenter Recommendation in Pull-Based Development [28]	-	Heuristic
[P18]	EARec: Leveraging Expertise and Authority for Pull-Request Reviewer Recommendation in GitHub [29]	EARec	Heuristic
[P19]	Core-reviewer recommendation based on Pull Request topic model and collaborator social network [30]	NTCRA	ML
[P20]	RSTrace+: Reviewer suggestion using software artifact traceability graphs [31]	RSTrace+	Heuristic
[P21]	A Large-Scale Study on Source Code Reviewer Recommendation [32]	-	ML
[P22]	Automatically Recommending Code Reviewers Based on Their Expertise: An Empirical Comparison [33]	WRC	Heuristic
[P23]	Who should make decision on this pull request? Analyzing time-decaying relationships and file similarities for integrator prediction [34]	TRFPre	ML
[P24]	WhoDo: Automating Reviewer Suggestions at Scale [35]	WhoDo	Heuristic
[P25]	Workload-Aware Reviewer Recommendation using a Multi-objective Search-Based Approach [36]	WLRRec	ML
[P26]	Using a Context-Aware Approach to Recommend Code Reviewers: Findings from an Industrial Case Study [37]	Carrot	ML
[P27]	Multi-objective code reviewer recommendations: balancing expertise, availability and collaborations [38]	-	ML
[P28]	Mitigating Turnover with Code Review Recommendation: Balancing Expertise, Workload, and Knowledge Distribution [39]	Sofia	Heuristic
[P29]	Expanding the Number of Reviewers in Open-Source Projects by Recommending Appropriate Developers [40]	-	ML



(a) Dataset Types (b) Dataset Sources

Fig. 4. Dataset characteristics. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

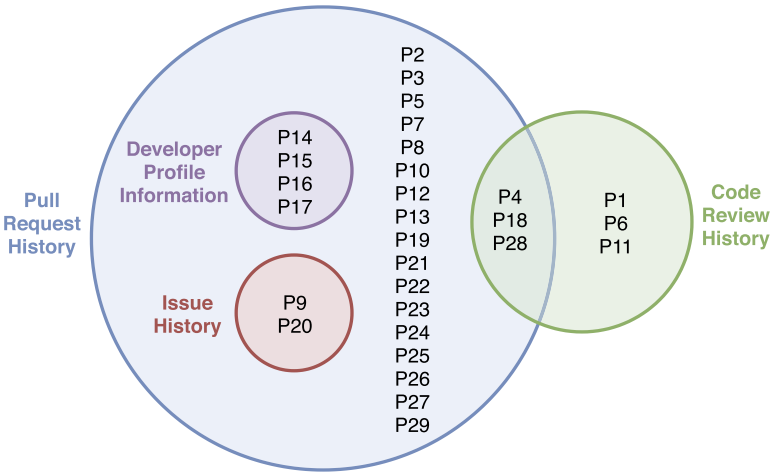


Fig. 5. Artifact types. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Researchers use different sources to collect data such as *GitHub*[8] and *Gerrit*[9] while mining project histories. Most of the studies (25 out of 29) use either GitHub or Gerrit as their data source. For GitHub repositories, they use the GitHub API[10] and GHTorrent [41]. Also, two studies use data from *Bugzilla*[11] while mining Mozilla projects. The distribution of dataset sources is shown in Fig. 4b. The *other* category includes the studies using a data source that is not used by another study. For example, the datasets of closed source projects are in the *other* category.

Note that the studies using the same dataset as previous CRR studies are considered as separate instances while counting the dataset resources and types. For example, one common dataset used in two different studies affects the statistics in Fig. 4b twice, thus we treat these two studies as separate instances because they use the dataset independently. Since one study may use multiple sources, the sum of the histogram bars in Fig. 4b is larger than the total number of studies investigated.

3.2.2. Artifact characteristics

The datasets used in CRR studies are created by mining different types of software artifacts such as pull requests and code review history. Fig. 5 presents the software artifacts used in corresponding CRR studies. *Pull request history* includes any kind of data coming from the change histories such as changed lines, paths of changed files, title and description. *Code review history* is related to the review process, and it includes comments made for pull requests and reviews. *Developer profile information* refers to features such as activeness and social features like the number of followers on GitHub. Lastly, *issue history* involves any textual data from issues such as the reporter, priority and severity information.

Pull request history is used by 26 studies while six studies use code review history, four studies use developer profile information and only two studies use issue history. All studies that use developer profile information or issue history also use pull request history. Besides, three studies that use code review history also use pull request history. Therefore, a total of nine studies uses two different types of artifacts, while none of the studies use three or four types of software artifacts. From a different viewpoint, the union of the studies using pull request history and code review history corresponds to all 29 primary studies while none of the studies use developer profile information or issue history alone.

3.3. Characteristics of evaluation setups (RQ3)

The evaluation setup for each study differs by the evaluation metrics and the validation techniques. Different metrics and validation techniques are discussed in the following sections. An overview of the evaluation setups used in CRR studies is presented in Table 7.

3.3.1. Evaluation metrics

Evaluation of a proposed method is a crucial part of measuring the degree of success of a specific approach. Evaluation metrics are used for comparing the results of different algorithms on the same datasets or comparing the results of an algorithm on different datasets. First, we formally define different and widely used evaluation metrics used in CRR studies.

[8] https://github.com/.
[9] https://www.gerritcodereview.com/.
[10] https://developer.github.com/.
[11] https://www.bugzilla.org/.

Table 7
Details of the Evaluation Setups of CRR Studies. (Metric Abbreviations: MP: Mean Precision, MR: Mean Recall, MFS: Mean F-Score, MRR: Mean Reciprocal Rank, Acc: Top-k Accuracy) (Validation Abbreviations: B-by-B: Batch by Batch, 1-by-1: One by One, OSW: Overlapping Sliding Window, N-OSW: Non-Overlapping Sliding Window, FTTS: Fixed Training & Test Sets).

Study ID	Metric(s) used	Validation approaches	Statistical tests	Running time evaluation	Questionnaire/Feedback/Interview	k values used in evaluation setups
[P1]	MP, MR	B-by-B				1, 2, 3, 4, 5
[P2]	MP, MR, MRR, Acc	OSW	✓			1, 3, 5
[P3]	Acc	1-by-1			✓	1, 2, 3, 4, 5
[P4]	MP, MR, MRR, Other	-	✓			1, 3, 5, 10
[P5]	Acc	1-by-1				1, 2, 3, 4, 5, 6, 7, 8, 9, 10
[P6]	MP, MR, MFS, MRR	1-by-1	✓			1, 2, 3, 5
[P7]	MRR, Acc	1-by-1				1, 3, 5, 10
[P8]	MP, MR	B-by-B				5, 10
[P9]	Acc	B-by-B	✓			1, 2, 3, 4, 5
[P10]	MRR, Acc	1-by-1				1, 3, 5, 10
[P11]	MP, MR, MFS	FTTS	✓			1, 2, 3, 4, 5, 6, 7, 8, 9, 10
[P12]	MP, MR, MFS, MRR	1-by-1	✓	✓		1, 2, 3, 4, 5, 6, 7, 8, 9, 10
[P13]	Acc	FTTS	✓			1, 3, 5, 10
[P14]	MRR, Acc	OSW	✓			1, 3, 5
[P15]	MRR, Acc	B-by-B	✓	✓		1, 2, 3, 4, 5
[P16]	MRR, Acc, Other	N-OSW				1, 3, 5
[P17]	MP, MR	1-by-1				1, 2, 3, 4, 5
[P18]	MP, MR, MFS	FTTS				1, 2, 3, 4, 5
[P19]	MP, MR	FTTS		✓		1
[P20]	MRR, Acc	1-by-1				1, 3, 5
[P21]	MRR, Acc	B-by-B	✓			1, 3
[P22]	Acc	1-by-1	✓			1, 3, 5
[P23]	MRR, Acc	B-by-B	✓	✓		1, 2
[P24]	MP, MR, MFS, Acc, Other	-			✓	1, 3
[P25]	MP, MR, MFS, Other	FTTS	✓			-
[P26]	MRR, Other	FTTS			✓	5
[P27]	MP, MR, MRR	-	✓		✓	1, 3, 5, 10
[P28]	MRR, Other	-			✓	-
[P29]	MRR, Acc	B-by-B				1, 3, 5, 10

Precision@k, *Recall@k* and *F-Score@k* are calculated as follows, where k stands for the number of recommended reviewers, r stands for the review, AR stands for the actual reviewers and RR stands for the recommended reviewers.

$$Precision@k(r) = \frac{|AR_r \cap RR_r|}{|RR_r|} \tag{1}$$

$$Recall@k(r) = \frac{|AR_r \cap RR_r|}{|AR_r|} \tag{2}$$

$$F\text{-}Score@k(r) = 2 \times \frac{Precision@k(r) \times Recall@k(r)}{Precision@k(r) + Recall@k(r)} \tag{3}$$

Mean Precision@k, *Mean Recall@k* and *Mean F-Score@k* are calculated as follows where R is the set of reviews.

$$Mean\ Precision@k(r) = \frac{1}{|R|} \sum_{r \in R} Precision@k(r) \tag{4}$$

$$Mean\ Recall@k(r) = \frac{1}{|R|} \sum_{r \in R} Recall@k(r) \tag{5}$$

$$Mean\ F\text{-}Score@k(r) = \frac{1}{|R|} \sum_{r \in R} F - Score@k(r) \tag{6}$$

Mean Reciprocal Rank (MRR) is calculated as follows where *rank(r)* returns the rank of the first correct reviewer in the ranked list of recommended reviewers, and if none of them is correct, it returns ∞ to make the reciprocal zero (0).

$$MRR(R) = \frac{1}{|R|} \sum_{r \in R} \frac{1}{rank(r)} \tag{7}$$

Top-k Accuracy is calculated as follows where *isCorrect(r)* returns 1 if any top-k recommended reviewer is an actual reviewer, otherwise returns 0.

$$Top\text{-}k\ Accuracy = \frac{1}{|R|} \sum_{r \in R} isCorrect(r) \tag{8}$$

Table 8
Studies grouped by evaluation metrics.

Evaluation metrics	Number of studies	Studies
Mean precision	13	P1, P2, P4, P6, P8, P11, P12, P17, P18, P19, P24, P25, P27
Mean recall	13	P1, P2, P4, P6, P8, P11, P12, P17, P18, P19, P24, P25, P27
Mean F-score	6	P6, P11, P12, P18, P24, P25
Mean reciprocal rank	16	P2, P4, P6, P7, P10, P12, P14, P15, P16, P20, P21, P23, P26, P27, P28, P29
Top-*k* accuracy	16	P2, P3, P5, P7, P9, P10, P13, P14, P15, P16, P20, P21, P22, P23, P24, P29
Other	6	P4, P16, P24, P25, P26, P28

In eight studies [P1, P6, P14, P15, P17, P23, P25, P28], researchers prefer calculating gain/improvement while others share the results in terms of the metrics above. Generally, $GainX_{A-B}$ is calculated as follows where A and B stand for different approaches and X stands for an evaluation metric.

$$GainX_{A-B} = \frac{X_A - X_B}{X_B} \tag{9}$$

Table 7 and Table 8 show the evaluation metrics defined above and the studies in which they are used. Since $GainX_{A-B}$ does not measure the performance directly, we do not include it in the figures.

Top-k accuracy (Equation (8)) and MRR (Equation (7)) are the most prevalent metrics with 16 studies using them. 13 studies use mean precision (Equation (4)) and mean recall (Equation (5)) at the same time, and six of them additionally use the mean F-score (Equation (6)). Mean reciprocal rank (Equation (7)) is used by 12 CRR studies. The *other* category includes the studies that define unique evaluation metrics and evaluate their approaches according to these metrics. Only six studies use such metrics in their evaluation. Each study uses either mean precision & recall or top-k accuracy.

Another factor about evaluation metrics is which k values are used while calculating the metrics above. Preferred k values in CRR studies are shown in Table 7. The most popular k values are 1, 3 and 5 since 21 studies use them in their evaluation setups.

Also, statistical tests are important while claiming that a recommendation model is significantly better than another one. As shown in Table 7, nearly half of the primary studies (14 studies) employ a statistical test in their evaluation.

Besides effectiveness, efficiency is another concern. As given in Table 7, only four studies [P12, P15, P19, P23] evaluate their methods in terms of training or test time.

In addition to evaluating the model on datasets, asking developers how well the models work in real life and receiving feedback from them is another way of evaluation. As shown in Table 7, five studies [P3, P24, P26, P27, P28] received feedback from developers while constructing their model or measuring the success of their model.

3.3.2. Validation techniques

The majority of CRR studies do not share detailed information about their validation techniques. In this section, by using explicit and implicit statements in the studies, we summarize the validation techniques used in CRR studies.

We categorize the different validation techniques used in CRR studies as *incremental training set*, *sliding window* and *fixed training-test set*. A common trait of all setups is that they use their dataset chronologically because all the studies use temporal data. Fig. 6 shows a visualization of the validation techniques and summarizes them in one figure.

The *incremental training set* category includes the approaches that increment the number of training instances in every step by starting from a definitive starting point and using some constant number of instances after the ending point as their test set.

There are two different *incremental training set* approaches depending on the increment size of the training set. As its name signifies, *one-by-one validation* tests data instances one by one while the model is trained by all the previous ones. After each step, the model is updated by using the last tested instance, then the new model is used for testing the next instance. One-by-one validation does not have to start from the beginning of the dataset. Some studies start to evaluate their model after some point. For example, Hannebauer et al. [P22] start to test one by one after training the model with one year of data. *Batch-by-batch validation* is similar to the one-by-one approach, but it uses a batch of instances instead of a single instance.

Another validation technique is *sliding window* validation. In the sliding window approach, training & test sets have fixed sizes, but their starting points slide over time. In other words, this technique slides both the starting and ending points of the training set and uses some constant number of instances after the ending point as the test set. We categorize sliding window approaches used in CRR studies as *overlapping sliding window* and *non-overlapping sliding window*.

As shown in Fig. 6, the overlapping window approach uses a fixed size window to train the model and tests the model with the instance next to the window. For the next iteration, the training set is slid by one instance, thus the training sets of two different iterations overlap. In the non-overlapping window approach, there is no overlap between training sets of two different iterations. Such an approach uses the test set of the previous iteration as the training set of the current iteration.

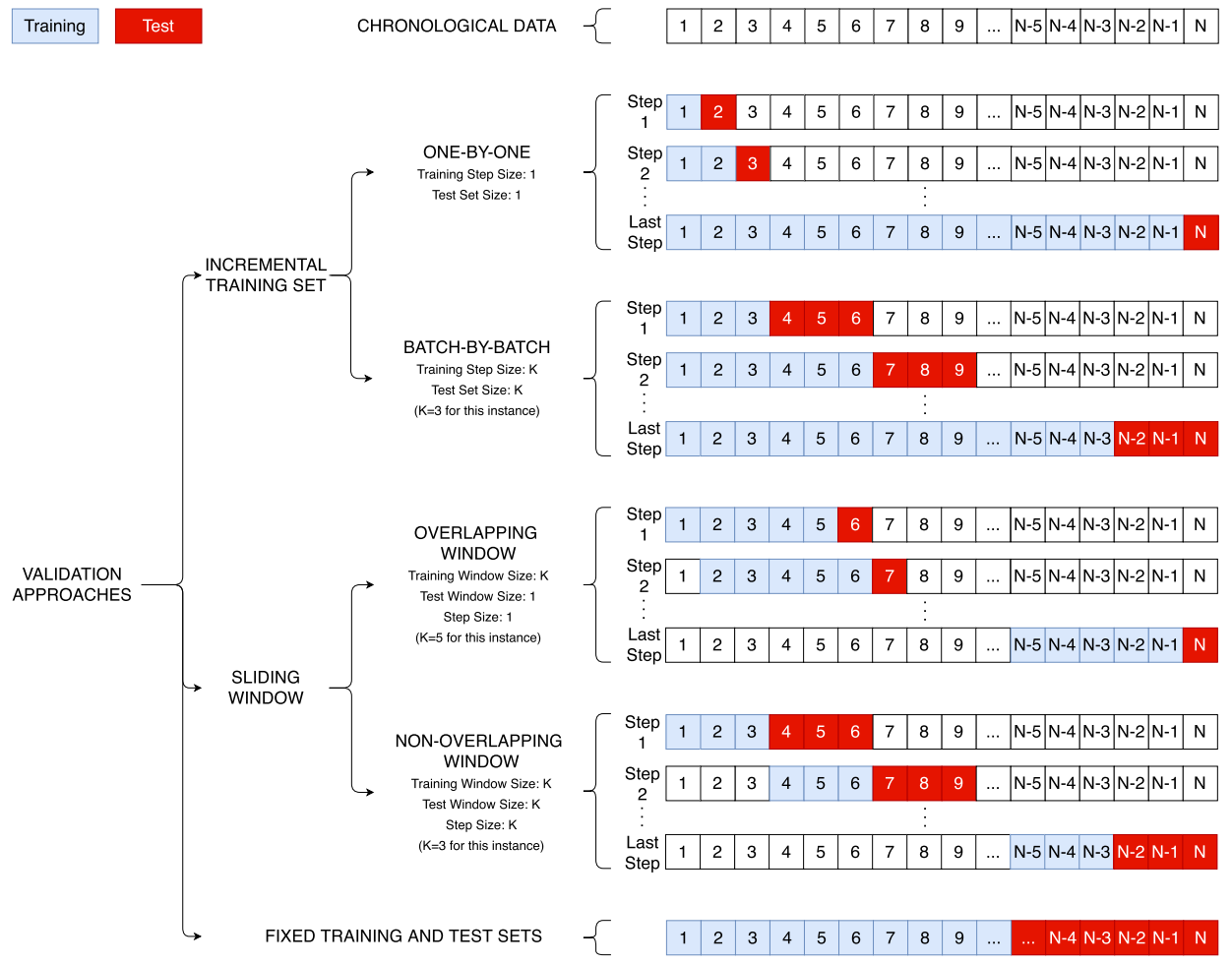


Fig. 6. Validation techniques. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Table 9
Studies grouped by validation techniques.

Experiment setup	Number of studies	Studies
One-by-one incremental training set	9	P3, P5, P6, P7, P10, P12, P17, P20, P22
Batch-by-batch incremental training set	7	P1, P8, P9, P15, P21, P23, P29
Overlapping sliding window	2	P2, P14
Non-overlapping sliding window	1	P16
Fixed train & test set	6	P11, P13, P18, P19, P25, P26
Not mentioned	4	P4, P24, P27, P28

Lastly, some of the studies split their datasets into two fixed parts with definitive starting and ending points for both the training & test sets as shown in Fig. 6. They train their model with the training set and test the model with the test set. Regardless of the testing approach such as one by one and whole test set at once, if the training & test sets are fixed-sized, then we collect these studies under the category of *fixed training & test sets validation*.

As illustrated in this section, there is a wide variety of evaluation setups used in CRR studies. This variety mostly arises from the differences in metrics used and the way that models are validated. Table 9 briefly shows the validation techniques and the studies using them.

3.4. Reproducibility of the studies (RQ4)

Gundersen and Kjensmo [42] propose three factors to enable reproducibility in machine learning studies: *experiment*, *data* and *method*. We investigate experimental validation techniques separately in Section 3.3.2. To assess the potential

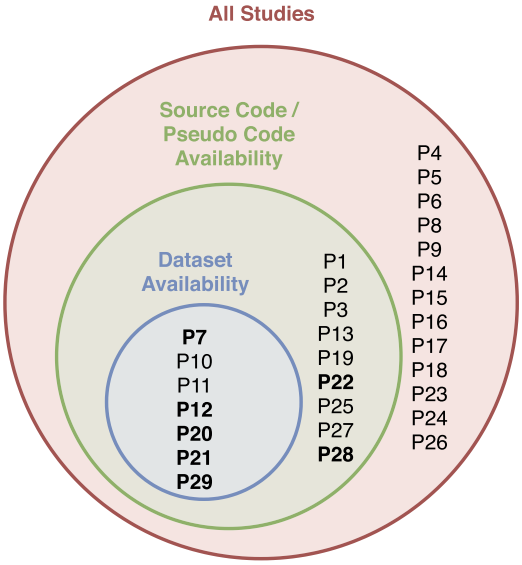


Fig. 7. Reproducibility criteria. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Table 10
Threats to validity discussions.

Threat category	Studies
Generalizability	P1, P2, P4, P6, P7, P9, P10, P11, P13, P14, P15, P17, P18, P20, P21, P22, P23, P25, P26, P27, P28, P29
Dataset integrity and validity	P1, P5, P6, P7, P9, P10, P11, P14, P15, P18, P20, P21, P22, P23, P25, P26, P29
Suitability of evaluation metrics	P2, P10, P15, P17, P20, P21, P23, P29
Implementation errors	P10, P14, P15, P20, P21, P23, P28
False recommendations with limited data	P2, P13, P19, P20
Dominance of active developers	P11, P15, P19, P21
Data collection process	P15, P20, P21, P23, P26
Missing/wrong features	P4, P11, P23, P26, P27, P28
Project dependency	P11, P18
Data dependency	P6
Other	P4, P6, P14, P27, P28
Not mentioned	P3, P8, P12, P16, P24

reproducibility of the studies, we define two reproducibility criteria: dataset availability (similar to *data* in [42]) and source code/pseudo code availability (similar to *method* in [42]). If both are available for a given study, one can often reproduce the results with ease.

Dataset availability refers to sharing datasets online for reproducing the study. Simply mentioning the data source and the starting and ending dates of the dataset may not be enough, because researchers may need exactly the same dataset for a proper comparison. The data may change over time or the data source may not be available anymore. Also, different studies might have different preprocessing steps, and they might not share a precise description of these steps. Therefore, we excluded the studies with dead web links and the studies sharing only the start and end dates. We only included the studies sharing their datasets online.

Source code/pseudo code availability means that the source code of the algorithm is shared online or the pseudo code is given in the study. We combine source code availability and pseudo code availability because these two criteria serve the same purpose. The studies that shared source code online [P7, P12, P20, P21, P22, P28, P29] are given in bold text in Fig. 7.

We summarize the reproducibility status of the studies in Fig. 7. As it can be seen from the figure, only seven studies meet both of our reproducibility criteria whereas nine studies only meet one criterion and 13 of the studies do not meet any criteria. Also, each study with source code or pseudo code also has an available dataset.

3.5. Threats to validity discussions (RQ5)

In Table 10, we summarized different types of threats to validity discussions that were mentioned in the CRR primary studies.

Generalizability refers to the extension of research results on different datasets. For example, Yu et al. [P11] mention that their findings are based on open source projects from GitHub and that it is unknown if their results can be generalized to

Table 11
Future work discussions.

Categories	Future works	Studies
Extending the evaluation	Experiments on (more) OSS projects	P4, P20
	Experiments on (more) closed source projects	P4, P9, P20
	Experiments on other projects	P7, P8, P10, P13, P15, P16, P17, P18, P23, P26, P27, P28, P29
Refining the model	Project specific models	P1
	Hybrid models	P11, P13, P22
	Applying/updating (more) models	P2, P4, P11, P12, P15, P17, P25, P27, P28
	Adding more software artifacts	P6, P17, P18, P23
	Considering more technical factors	P4, P11, P16, P17, P18, P20, P22
	Considering more non-technical factors	P4, P11, P12
	Considering more factors	P5, P23, P24, P25, P27, P28
Tool support	Available tool support	P7, P14, P26
Not mentioned	No future work discussion	P3, P19, P21

the open source projects hosted on coding platforms other than GitHub or closed source projects. Generalizability is the most popular threat to validity as it is discussed by 22 out of 29 studies.

Dataset integrity and validity includes any kind of discussion about the completeness and correctness of the datasets. Missing information in datasets such as the retirement of developers [P7, P9] and availability of developers [P18] are good examples of integrity problems. Also, the uncertainty about whether the real-life reviewers are the correct reviewers or not is one of the major discussions about dataset validity [P1, P4, P14].

Limitations of evaluation metrics fall under *the suitability of evaluation metrics* category. For instance, Lipcak and Rossi [P21] discuss that they only use the top-k accuracy and MRR metrics because of their popularity.

Implementation errors are simply the bugs which may be present in the source code of the CRR algorithms. The authors discuss the implementation issues of their own methods [e.g., P10, P20] and the compared methods [e.g., P14].

False recommendations with limited data refer to the discussions regarding algorithms that do not perform well with a limited amount of data, especially at the initial few pull requests. For instance, graph methods have problems with recommending the correct code reviewers for the initial pull requests in a project [P20].

Discussions about the *dominance of active developers* mention that the algorithms tend to recommend the most active developers in the projects [P11, P15, P19]. This is also a problem for the workload of the developers [P21].

Data collection process refers to the problems during the mining process of the dataset from a platform such as GitHub. For example, developers may not be identified correctly because they can be registered with different names to different platforms or they can use multiple usernames on the same platform [P23].

Missing/wrong features category refers to the discussions about unnoticed and unstudied features. For instance, Jiang et al. [P23] mention that they use only decision history but there may be other communication channels between integrators and contributors.

Some authors discuss that a part of their approaches [P11] or some parameters in their models [P18] depend on the project to which they apply their recommendation model, thus we mention these studies under *project dependency*. Also, *data dependency* refers to the discussion about the issue that different historical periods might produce different results [P6]. *Other* refers to the discussions that do not fit any of the categories above. For instance, Zanjani, Kagdi and Bird [P6] mention that ignoring the roles of reviewers could potentially create a threat to validity. Lastly, some studies [P3, P8, P12, P16, P24] do not discuss their threats to validity or limitations. Therefore, we collect them in the *not mentioned* category.

3.6. Future work discussions (RQ6)

Future work discussions can be summarized under four categories which are *extending the evaluation*, *refining the model*, *tool support* and *not mentioned* as displayed in Table 11.

Extending the evaluation category includes any kind of extra experiments on open source and closed source projects. Many studies do not specifically mention that they plan to make additional experiments on open or closed source projects, thus we assign them to the experiments on other projects category. For example, Thongtanunam et al. [P7] discuss that their approach needs additional evaluation with other datasets including a larger number of reviewers to better generalize the results.

Refining the model category refers to any kind of improvement or fix for the models. One study [P1] proposes project-specific models for future work. Some studies [e.g., P11, P15] propose applying new learning/heuristic models or updating the existing models. A few studies [P11, P13, P22] propose to combine different algorithms and produce hybrid methods. A group of studies [P4, P6, P17, P18, P23] plan to use more software artifacts in their model and datasets to improve their methods.

Also, 14 out of 29 studies discuss considering more factors. Some of them specifically emphasize technical factors [e.g., P11, P16] and non-technical factors [P4, P11, P12], but others do not [P5, P23, P24]. Thus, we collect them under the

Table 12
Summary of the discussions from different perspectives.

Perspectives	Prominent discussion points
Implications for software engineering research community	Heuristic approaches are mostly preferred. Recently, machine learning approaches have started to become prominent. Making additional experiments and refining the models by considering more software artifacts or factors are the most popular future work discussions. Providing tool support is a rare but important future discussion. Research on scalability and maintainability of the algorithms are another future direction.
Implications for software engineering practice	More validation in closed source environments is needed. Due to their different nature, the models using open source data might not work in closed source projects. Collaboration between academia and industry might help improving the models. Validating the models with developer feedback is needed to assess the applicability of CRR systems.
Prevalent differences	CRR terminology differs among the studies. Different evaluation metrics are preferred in different studies. There is no one common metric for all. Validation techniques are different among the studies, and it makes comparison more difficult.
Reproducibility	In general, studies suffer from reproducibility issues. Future researchers should be more open to share more details.
Data integrity and validity	The studies suffer from ground truth validation and the noise in the data. Future researchers should provide more convenient techniques to overcome such problems and be open to share more details.

considering more factors category. For instance, Tang et al. [P5] discuss that adding or changing some features of their model might improve the results.

Future work about providing available tool support is under the *tool support* category [P7, P14, P26].

Lastly, three studies [P3, P19, P21] do not share any proposal for future work. Therefore, we collect them under the *not mentioned* category.

4. Discussion

This section reveals the implications of our study for the software engineering research community & practice, prevalent differences in CRR studies, followed by reproducibility and data validity problems of the studies. We summarize the key takeaway messages in Table 12.

4.1. Implications for software engineering research community

Since the authors of the most of the studies (23 out of 29) are only affiliated to a university, it is clear that academia leads CRR research. Our study provides insights from different perspectives, and we compiled four suggested perspectives for improving CRR research: types of methods used, discussed future works, run-time evaluation of the methods and maintenance of the model.

First, the types of the methods used are important to see which approaches were utilized by the research community to solve the CRR problem. It is clear that heuristic approaches are slightly more popular in CRR studies (see Section 3.1).

Initial CRR studies focused on heuristic models, however, a few researchers [P9, P14] started to use their knowledge for feature extraction and let the machine learning models learn the relations instead of combining the features with some intuitive mathematical equations. Recently, machine learning approaches have started to become prominent, but still, the heuristic approaches are mostly preferred because they are more explainable than the machine learning approaches (see Section 3.1). Also, a few studies [P5, P10] try to use the advantages of both heuristics and machine learning techniques by combining their results in hybrid approaches.

Related to the possible research directions for CRR studies, future work discussions are mostly about making additional experiments and refining the models by considering more software artifacts or factors (see Section 3.6). One rare but important discussion about future work is providing a CRR tool for version control platforms such as GitHub. Using tool support in a real-life project could potentially be a better validation technique compared to evaluation on datasets, as getting feedback from developers using a CRR tool is more informative than considering just the history of the dataset by using metrics such as top-k accuracy and MRR. Only five primary studies [P2, P3, P20, P26, P28] provided tool support for their proposed recommendation system.

Another future direction would be the evaluation of the algorithms in terms of training & test time. Proposing an algorithm without considering its time complexity and resource efficiency may not be realistic for real-life cases. Only five of the CRR studies [P12, P15, P19, 20, P23] shared some results indicating how long the experiments took in terms of training & test time.

Lastly, future studies can focus on discussing the potential costs of maintaining CRR methods over time. For example, updating machine learning models when a new pull request comes is potentially time-consuming. The same issue is also valid for graph-based approaches. Graph approaches such as [P20] initially seem like they are open to change, and they

can be updated for every new accepted pull request. Yet the graph will be larger over time, meaning that predicting a new reviewer will take longer. This will raise the issue of the evaluation of run time discussed in the previous paragraph.

4.2. Implications for industry/software engineering practice

Related to the implications for industry and software engineering practice, we identified five main discussion points from the results obtained in Section 3: the types of datasets used, the artifacts used, the practical value of CRR, the perceived importance of CRR and the appropriateness of evaluation.

First, the dataset types used in recommendation models are important indicators of their practical implications. As stated in Section 3.2.1, 79.3% (23 out of 29) of the primary studies investigated in this systematic review use only open source code reviewer data. As the dynamics of the open source community and the industry might potentially differ, what works for open source projects might not work in the industry or vice versa. Bosu et al. [43] revealed that the code review process shows some differences among Microsoft employees and open source software contributors in different aspects. Related to this, 22 studies out of 29 claim that their models might not be generalizable to other OSS/industry projects. Also, 16 studies share more "experiments on other projects" as a future direction. Three of them [P4, P9 and P20] specifically aim to extend their experiments on industrial projects and get feedback from developers. [P14] shares the same concern in the following way: "*Our results were obtained with the use of open-source project data. Therefore, we cannot ensure that the results will be the same for industrial projects*." In the future, more validation in closed source environments is needed. Active collaboration between academia and industry would potentially enhance the real-life impact of recommendation models/algorithms.

Related to using different data sources, the recent studies [P4, P25, P26, P27, P28] take the workload of the developers into account while designing their model. Also, two studies [P22, P24] stated workload (developer/reviewer availability) inspection as their future work and four studies [P7, P15, P18, P21] discussed the workload of the developers as a threat to their validity. Considering the workload of the developers is necessary for real-life applicability of the CRR models because the best candidate reviewers are not the best choices if they are not available for the review.

Second, there is a range of artifacts used in the studies such as pull request history, code review comment history, issue history and developer profile data. Among these four artifact types, all except developer profile data are very popular and accessible in both open source and proprietary software projects. It might be infeasible to access developer profile data in other organizations since some companies do not have the required structure to record developer activity as in GitHub. Using a diverse set of artifacts as input to the models might enhance the recommendation success.

Third, the practical value of reviewer recommendation can be observed by some examples from real life. Many large software companies such as IBM [44], Microsoft [45] and VMware [46] have patented their code reviewer recommendation models/tools. Beyond these patents, in the recent years, many code review platforms started to offer an automated reviewer recommendation feature. Upsource suggests code reviewers based on the history of changed files and code review history [9]. Crucible recommends reviewers by analyzing the developer contributions to the selected files [10]. Gerrit allows to customize the reviewer recommendation method by providing a ranking model [8]. GitHub provides two different code reviewer recommendation algorithms to be used within software development teams [11]: round robin and load balance. Beyond that, there are some other tool integrations to assign appropriate code reviewers for a pull-request such as Pull Approve[12] and Pull Assigner.[13]

Fourth, the perceived importance of CRR models/tools has been discussed in the literature. Kovalenko et al. [47] performed a comprehensive study to discover the developers' perceptions on code reviewer recommendation tools. They conducted surveys and interviews among JetBrains and Microsoft developers. Their results indicated that the majority of developers (56% for Microsoft and 69% for JetBrains) found the reviewer suggestions of CRR tools relevant. However in another survey question, the developers were asked whether the reviewer recommendations of the tools were helpful for them or not. 69% of the developers in JetBrains responded to this question as *Never*. For the same question, 45% of the Microsoft developers stated that they found the reviewer suggestions of the tools helpful occasionally or less often. The interviewees explained this situation such that they already knew who to assign the changeset before the tool recommendation. Additionally, the authors claimed that the context of a code review might differ occasionally (i.e., changing legacy code, introducing a new team member to the codebase). In this regard, future researchers might consider introducing more user-centric and context-aware CRR models to handle such problems.

Finally, the appropriateness of evaluation for code reviewer recommendation models/tools is an important discussion mentioned in some of the primary studies. Since the code reviewer assignment task can be affected by many non-technical factors (availability, workload, etc.) in real life, the quantitative evaluation by using only performance metrics (precision, recall, MRR, accuracy, etc.) might be inadequate. Rather than just relying on performance metrics only, the users of the reviewer recommendation tool should also be consulted to gather feedback related to the quality of the reviewer recommendations. Only a few primary studies mentioned this problem:

[12] https://www.pullapprove.com/.
[13] https://pullpanda.com/assigner.

- The authors of [P26] deployed their tool in one Ericsson project and evaluated it in terms of how successful the recommendations were from the point of view of the tool users and the recommended reviewers.
- Similarly in [P20], after each reviewer recommendation, the tool sends a survey link to the PR owner in order to assess the performance of the recommendation. Although they did not make a case study with this survey evaluation, they noted this aspect as a future direction.

4.3. Prevalent differences

Among the 29 CRR studies, we observed three significant prevalent differences: general CRR terminology, evaluation metrics and validation techniques.

Generally, studies use *code reviewer* or *reviewer* to describe recommended developers [P1, P2] for a code review, however some of the studies use peer reviewer [P4, P6], patch reviewer [P8], integrator [P14, P23], core member [P15], commenter [P17] and core-reviewer [P19]. Even though they use different terminology, all of them refer to nearly the same meaning.

Another difference is the evaluation metrics used in the CRR studies. A few of the studies [P22, P24] do not share details of the metrics. In general, CRR studies are divided into two groups according to the evaluation metrics that they use: those using a combination of mean precision & mean recall and those using a combination of top-k accuracy & MRR (see Section 3.3.1). Also, there are a few uncommon evaluation metrics not used in other CRR studies. For example, Ye [P16] uses Mean Average Precision, and Asthana et al. [P24] use metrics like pull request completion time to evaluate their method.

It is a challenging task to compare different recommendation models with different evaluation metrics. As a general principle, a new recommendation model should be compared with the previous ones in terms of the same evaluation metrics in order to prove the advantages of the study. Beyond this, we identified two more issues regarding the evaluation metrics:

- *Choosing k values:* Precision, recall and F-measure are calculated at a cutoff value, k. Lipcak and Rossi [P21] state that using these three metrics might not be appropriate since the number of the actual code reviewers might be less than k and the recommended set of code reviewers would include inevitable false positives. Also, Zanjani, Kagdi and Bird [P6] shared the number of code reviewers in their datasets (both open and closed source projects). Mostly, reviews include one (66%) or two reviewer(s) (24%). Differently than the open source community, in industry, different teams are responsible for specific sections of the source code. According to the Scrum Guide [48], an ideal software team size should be between three and nine people. In these types of scenarios, the code will be reviewed by one of the team members. If the k value is chosen to be a larger number than the team size, then the top-k evaluation results of a CRR study may become meaningless. Therefore, using precision, recall and F-measure with high k values would not be appropriate.
- *Ignoring the recommendation order:* Another concern is that while calculating precision, recall and accuracy metrics at k value, the ordering of the recommended reviewers is ignored. The metric of *mean reciprocal rank (MRR)* might be a good alternative to these metrics as it considers the order of the recommended results.

The last prevalent difference is the validation techniques used in the CRR studies. One-by-one validation is clearly the preferred method for many cases (see Section 3.3.2). Selecting a validation technique is not always about preferences; sometimes proposed CRR algorithms may require a specific validation type. For example, one-by-one validation might be an inconvenient option for some machine learning models, because they are more suitable to be used with fixed-size training & test sets. Similarly, some studies use overlapping and non-overlapping sliding windows for validation since these validation techniques are more proper for evaluating such models. Such a variety of validation setups makes comparison among CRR studies more difficult.

4.4. Reproducibility

Reproducibility is the ability of reproducing the same results by using the same data and method [42]. It is an important and desirable property in data-driven empirical software engineering studies [49]. Fernández and Passoth [50] state that empirical software engineering studies suffer from not being transparent and replicable due to the lack of shared data. Lately, the software engineering community has made some significant improvements in open science. The ROSE Festival[14] has been held in order to recognize and reward open science in software engineering studies. Similarly, Fernández et al. [51] published an open science initiative for Empirical Software Engineering journal.

Despite these efforts towards open science, many primary studies (13 of 29) do not meet any of our reproducibility criteria: dataset availability and source code/pseudo code availability (see Section 3.4). These criteria are important to reproduce results and compare new algorithms with the previous ones. Among the 29 studies investigated within this SLR, we have not observed an increasing pattern in the number of reproducible studies over the years. Generally, researchers

[14] https://2019.icse-conferences.org/track/icse-2019-ROSE-Festival.

tend to share the source code or the pseudo code of their algorithms, but they have not shared the datasets used in the experiments online, even though 23 out of 29 studies use only open source projects.

In order to increase transparency and reproducibility, future researchers in CRR studies need to be more open about sharing their models and datasets.

4.5. Data integrity and validity

In general, CRR studies suffer from two types of data related problems: *noise in the data* and *ground truth validity problems in the data*.

Noise in the data refers to measurement and labeling related errors in data, which occur in a random manner. There are certain types of noise in code reviewer data. In some cases, it is possible to eliminate or reduce the effect of the noise. For instance, the same developer might have different usernames on different platforms [P23] (e.g., *John D.* on Jira and *John Doe* on GitHub). Some preprocessing actions can be taken to reduce the effect of such noise in these datasets. On the other hand, there are some cases in which the noise in the data cannot be eliminated. For example, assigning an unintended reviewer for a pull request by mistake (e.g., *due to developers with similar/same names*) cannot be fixed.

Ground truth validity, on the other hand, is another important issue to clarify in data-driven empirical software engineering studies. Dogan et al. [52] investigates the ground truth problem in CRR studies. They claim that the assigned reviewers in real life were not necessarily the best possible reviewer for a given pull request. For this reason, recommendation models using real-life reviewer assignment instances from open source and proprietary software projects have a potential ground truth problem.

While 17 of 29 primary studies mention this problem as a threat to their validity, none of them propose a feasible solution approach for this problem (see Section 3.5). Future studies should take this issue into account and propose some ways to deal with the possible ground truth problems.

5. Threats to validity

Threats to validity can be defined as the possible aspects of the research design that might compromise the credibility of research results, and secondary studies are vulnerable to several threats to validity [53]. In this section, we adopted the guidelines of Ampatzoglou et al. [53] and Zhou et al. [54] on the threats to validity in software engineering secondary studies.

Within this study, we investigated three different types of validity: *construct validity, internal validity* and *external validity*.

Construct validity investigates the suitability of measures for the research study. In the case of systematic literature reviews, the main task is to find all related studies to a specific topic and answer the RQs by analyzing them. The threats to construct validity that we faced and the mitigations that we applied for them are listed below:

Selection of Digital Libraries: The selection of digital libraries to complete a search is a critical issue. Although it is not possible to check all digital libraries, we tried to solve this issue by including the most popular digital libraries and applying forward & backward snowballing.

Construction of the Search String: Before starting the detailed search process, the search string given in Section 2.2 is constructed by checking five reviewer recommendation studies. We tried to use synonyms/similar words of each part of the string (i.e., *code* reviewer, *pull request* reviewer, *patch* reviewer). Even if we tried to span all search queries by our string, we may have missed some CRR studies. To mitigate this problem, we applied forward and backward snowballing to significantly reduce the chance of missing a study.

It is not possible to use the same string structure for different digital libraries. Each of them has different constraints on the search string (e.g., the number of Boolean operators). To mitigate this threat, we made little modifications and shortenings on the search string so that it could be used in each of digital libraries. However, the general structure of each search string is similar to the one given in Section 2.2.

Research Questions: Research questions are defined such that a comprehensive analysis of CRR studies can be achieved. It is a crucial threat to miss some important research questions. To mitigate this threat, all primary studies were investigated by the authors in detail to not miss any important detail to be questioned by RQs.

Defining the Time Interval of Primary Studies: As there is no previous SLR on CRR studies available in the literature, we did not explicitly state a starting publication date and covered all primary studies published before November 2020.

Internal validity for a systematic literature review is the rigorous implementation of the SLR process.

To minimize the risk of any subjective activity in the SLR process, we followed the SLR guidelines of Kitchenham and Charters [12], and we applied a triple-check approach. While selecting the primary studies and extracting data from them, all steps were completed by two of the authors, independently. When there was any conflict, the third author was involved in the decision process.

External validity can be defined as the extent to which the effects of a study can be generalized to outside of the study [12]. This systematic review is conducted by following the SLR guidelines of Kitchenham and Charters [12], and it is

repeatable by following the steps in Section 2. All data extracted from the 29 primary studies is available online.[15] The data extraction process and results obtained are accurate only for the CRR domain. Any missing or future study can easily be analyzed with respect to our methodology.

6. Related work

To the best of our knowledge, there has been no SLR study on CRR. On the other hand, a systematic mapping study of modern code review [55] has been published besides a few empirical comparison studies [32,33] of CRR algorithms. In the following, we summarize the related work and provide comparisons with our SLR study.

Badampudi et al. [55] shared the preliminary results of their systematic mapping study on modern code review (MCR). They specified different aspects (impact and outcome of code review, proposed solutions and reviewer selection criteria) of MCR practice over time. They identified 177 research studies between 2005 and 2018. These studies are all somehow related to different aspects of modern code review, but not specifically related to the CRR problem. Since the study shared the preliminary results, it just mentioned reviewer selection studies, thus a comprehensive comparison of different CRR methods is not available. In the study, they claimed that they found 23 studies about reviewer selection but they referenced 19 of them in the corresponding section. 14 of these 19 studies intersect with our studies. The remaining five studies are the prior articles of others. (We only considered the latest articles of CRR models as mentioned in Section 2.3.) In our study, we provided a comprehensive analysis of 29 primary CRR studies published from 2009 to 2020.

Hannebauer et al. [33] performed an empirical comparison of expertise-based CRR algorithms. They compared the prediction performance of eight CRR approaches by using four open source projects. One of the algorithms is the File Path Similarity (FPS) [7] approach, another one is a simplified version of FPS, and the other six approaches are based on modification expertise metrics that evaluate the expertise of a software developer related to a software artifact. Since it proposed to use different expertise metrics, it is among our primary studies. Even though it is a comparison study, they did not perform a systematic review of the literature for CRR studies and did not discuss other CRR models as we did in our study.

Similarly, Lipcak and Rossi [32] analyzed and compared six previous CRR models besides their Naive Bayes-based method. They did not review the literature systematically for CRR models, and they did not share an in-depth discussion for all CRR studies up to the present. While they compared seven algorithms in total, our study investigated 29 CRR papers.

7. Conclusion

Code review, one of the most popular practices in software development, plays a crucial role in software development. One of the key tasks in code review practice is the proper and automated selection of the code reviewer(s). In the recent years, popular tools & platforms such as GitHub, Gerrit, Upsource and Crucible started to offer an automatic reviewer recommendation feature, indicating the growing interest of this field. Also, large software companies such as IBM [44], Microsoft [45] and VMware [46] have patented their code reviewer recommendation models/tools.

Since 2009, especially in the last six years, several studies proposed CRR approaches to address this problem. In this study, we performed a systematic literature review on code reviewer recommendation studies. As of November 2020, we identified 29 primary studies related to the subject. With six research questions, we investigated code reviewer recommendation studies from various perspectives.

We summarize some of the key findings of our study as follows:

- Heuristic approaches are slightly more popular (14 out of 29, 48%), but machine learning techniques have started to become popular in recent years (13 out of 29, 45%).
- Most studies suffer from reproducibility issues. 13 studies (45%) meet none of our reproducibility criteria, while only seven studies (24%) meet all criteria: *dataset availability and source code/pseudo code availability*.
- Generalizability of the model, dataset integrity and suitability of evaluation metrics are the main concerns for threats to validity in CRR studies.
- The majority of the studies (23 out of 29, 79%) evaluate their model on only open source projects.
- There are prevalent differences among studies such as terminology, evaluation metrics and validation setup which complicate the comparison among these studies.
- Refining models and conducting additional experiments on open and closed source datasets are the most common future work discussed in the studies.

Ultimately, we hope that this literature review will lead researchers to a better understanding of the current state of code reviewer recommendation research.

[15] https://figshare.com/articles/dataset/Data_Extraction_Form/13439366.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported In this paper.

Appendix A. The list of primary studies

P#	Article title	Venue and year	Authors
P1	A hybrid approach to code reviewer recommendation with collaborative filtering [14]	International Workshop on Software Mining (2017)	Z. Xia, H. Sun, J. Jiang, X. Wang, X. Liu
P2	CoRReCT: Code reviewer recommendation in GitHub based on cross-project and technology experience [15]	International Conference on Software Engineering Companion (2016)	M.M. Rahman, C. K. Roy, J.A. Collins
P3	Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation [16]	International Conference on Software Engineering (2013)	V. Balachandran
P4	WhoReview: A multi-objective search-based approach for reviewers recommendation in modern code review [17]	Applied Soft Computing Journal (2020)	M. Chouchen, A. Ouni, M.W. Mkaouer, R.G. Kuka, K. Inoue
P5	RevRec: A two-layer reviewer recommendation algorithm in pull-based development model [18]	Journal of Central South University (2018)	C. Yang, X. Zhang, L. Zeng, Q. Fan, T. Wang, Y. Yu, G. Yin, H. Wang
P6	Automatically recommending peer reviewers in modern code review [19]	IEEE Transactions on Software Engineering (2015)	M.B. Zanjani, H. Kagdi, C. Bird
P7	Who should review my code? A file location-based code-reviewer recommendation approach for modern code review [7]	International Conference on Software Analysis, Evolution and Reengineering (2015)	P. Thongtanunam, C. Tantithamthavorn, R.G. Kula, N. Yoshida, H. Iida, K. Matsumoto
P8	Patch reviewer recommendation in OSS projects [20]	Asia-Pacific Software Engineering Conference (2013)	J. B. Lee, A. Ihara, A. Monden, K. Matsumoto
P9	Improving code review by predicting reviewers and acceptance of patches [13]	Research on Software Analysis and Error-free Computing (2009)	G. Jeong, S. Kim, T. Zimmermann, K. Yi
P10	Who should review this change?: Putting text and file location analyses together for more accurate recommendations [21]	International Conference on Software Maintenance and Evolution (2015)	X. Xia, D. Lo, X. Wang, X. Yang
P11	Reviewer recommendation for pull requests in GitHub: What can we learn from code review and bug assignment? [22]	Information and Software Technology (2016)	Y. Yu, H. Wang, G. Yin, T. Wang
P12	Profile based recommendation of code reviewers [23]	Journal of Intelligent Information Systems (2018)	M. Fejzer, P. Przymus, K. Stencel
P13	Understanding review expertise of developers: a reviewer recommendation approach based on latent Dirichlet allocation [24]	Symmetry (2018)	J. Kim, E. Lee
P14	Automatic assignment of integrators to pull requests: the importance of selecting appropriate attributes [25]	Journal of Systems and Software (2018)	M. L. de Lima Júnior, D. M. Soares, A. Plastino, L. Murta
P15	CoreDevRec: Automatic core member recommendation for contribution evaluation [26]	Journal of Computer Science and Technology (2015)	J. Jiang, J.H. He, X.Y. Chen
P16	Learning to rank reviewers for pull requests [27]	IEEE Access (2019)	X. Ye
P17	Who should comment on this pull request? Analyzing attributes for more accurate commenter recommendation in pull-based development [28]	Information and Software Technology (2017)	J. Jiang, Y. Yang, J. He, X. Blanb, L. Zhang
P18	EARec: Leveraging expertise and authority for pull request reviewer recommendation in GitHub [29]	International Workshop on Crowd Sourcing in Software Engineering (2016)	H. Ying, L. Chen, T. Liang, J. Wu
P19	Core-reviewer recommendation based on pull request topic model and collaborator social network [30]	Soft Computing (2019)	Z. Liao, Z. Wu, Y. Li, Y. Zhang, X. Fan, J. Wu
P20	RSTrace+: Reviewer suggestion using software artifact traceability graphs [31]	Information and Software Technology (2020)	E. Sülün, E. Tüzün, U. Doğrusöz

(continued on next page)

P#	Article title	Venue and year	Authors
P21	A large-scale study on source code reviewer recommendation [32]	Euromicro Conference on Software Engineering and Advanced Applications (2018)	J. Lipcak, B. Rossi
P22	Automatically recommending code reviewers based on their expertise: an empirical comparison [33]	International Conference on Automated Software Engineering (2016)	C. Hannebauer, M. Patalas, S. Stünkel
P23	Who should make decision on this pull request? Analyzing time-decaying relationships and file similarities for integrator prediction [34]	Journal of Systems and Software (2019)	J. Jiang, D. Lo, J. Zheng, X. Xia, Y. Yang, L. Zhang
P24	WhoDo: Automating reviewer suggestions at scale [35]	European Software Engineering Conference and Symposium on the Foundations of Software Engineering (2019)	S. Asthana, R. Kumar, R. Bhagwan, C. Bird, C. Bansal,C. Maddila, S. Mehta, B. Ashok
P25	Workload-Aware Reviewer Recommendation using a Multi-objective Search-Based Approach [36]	International Conference on Predictive Models and Data Analytics in Software Engineering (2020)	W. H. A. Al-Zubaidi, P. Thongtanunam, H. K. Dam, C. Tantithamthavorn, A. Ghose
P26	Using a Context-Aware Approach to Recommend Code Reviewers: Findings from an Industrial Case Study [37]	International Conference on Software Engineering: Software Engineering in Practice (2020)	A. Strand, M. Gunnarson, R. Britto, M. Usman
P27	Multi-objective code reviewer recommendations: balancing expertise, availability and collaborations [38]	Automated Software Engineering (2020)	S. Rebai, A. Amich, S. Molaei, M. Kessentini, R. Kazman
P28	Mitigating Turnover with Code Review Recommendation: Balancing Expertise, Workload, and Knowledge Distribution [39]	International Conference on Software Engineering (2020)	E. Mirsaeedi, P. C. Rigby
P29	Expanding the Number of Reviewers in Open-Source Projects by Recommending Appropriate Developers [40]	International Conference on Software Maintenance and Evolution (2020)	A. Chueshev, J. Lawall, R. Bendraou, T. Ziadi

References

[1] M.V. Mäntylä, C. Lassenius, What types of defects are really discovered in code reviews?, IEEE Trans. Softw. Eng. 35 (3) (2008) 430–448.
[2] M. Fagan, A history of software inspections, in: Software Pioneers, Springer, 2002, pp. 562–573.
[3] C. Sadowski, E. Söderberg, L. Church, M. Sipko, A. Bacchelli, Modern code review: a case study at Google, in: Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice, ACM, 2018, pp. 181–190.
[4] J. Klünder, R. Hebig, P. Tell, M. Kuhrmann, J. Nakatumba-Nabende, R. Heldal, S. Krusche, M. Fazal-Baqaie, M. Felderer, M.F.G. Bocco, et al., Catching up with method and process practice: an industry-informed baseline for researchers, in: Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, IEEE Press, 2019, pp. 255–264.
[5] L. MacLeod, M. Greiler, M.-A. Storey, C. Bird, J. Czerwonka, Code reviewing in the trenches: challenges and best practices, IEEE Softw. 35 (4) (2018) 34–42.
[6] A. Bacchelli, C. Bird, Expectations, outcomes, and challenges of modern code review, in: 2013 35th International Conference on Software Engineering, ICSE, IEEE, 2013, pp. 712–721.
[7] P. Thongtanunam, C. Tantithamthavorn, R.G. Kula, N. Yoshida, H. Iida, K.-i. Matsumoto, Who should review my code? a file location-based code-reviewer recommendation approach for modern code review, in: 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering, SANER, IEEE, 2015, pp. 141–150.
[8] Gerrit code review - plugin development, https://gerrit-review.googlesource.com/Documentation/dev-plugins.html#reviewer-suggestion. (Accessed 23 January 2021).
[9] Upsource :: Code review, project analytics :: Features, https://www.jetbrains.com/upsource/features/automatedcodereview.html. (Accessed 23 January 2021).
[10] Crucible - code review tool - choosing reviewers, https://confluence.atlassian.com/crucible/choosingreviewers-298977465.html. (Accessed 30 January 2021).
[11] Managing code review assignment for your team, https://docs.github.com/en/free-pro-team@latest/github/setting-up-and-managing-organizations-and-teams/managing-code-review-assignment-for-your-team. (Accessed 12 December 2020).
[12] B. Kitchenham, S. Charters, Guidelines for performing systematic literature reviews in software engineering, Technical report, Ver. 2.3 EBSE Technical Report. EBSE, 2007.
[13] G. Jeong, S. Kim, T. Zimmermann, K. Yi, Improving code review by predicting reviewers and acceptance of patches, 2009, pp. 1–18, Research on software analysis for error-free computing center Tech-Memo (ROSAEC MEMO 2009-006).
[14] Z. Xia, H. Sun, J. Jiang, X. Wang, X. Liu, A hybrid approach to code reviewer recommendation with collaborative filtering, in: 2017 6th International Workshop on Software Mining, SoftwareMining, IEEE, 2017, pp. 24–31.
[15] M.M. Rahman, C.K. Roy, J.A. Collins, Correct: code reviewer recommendation in GitHub based on cross-project and technology experience, in: 2016 IEEE/ACM 38th International Conference on Software Engineering Companion, ICSE-C, IEEE, 2016, pp. 222–231.
[16] V. Balachandran, Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation, in: Proceedings of the 2013 International Conference on Software Engineering, IEEE Press, 2013, pp. 931–940.
[17] M. Chouchen, A. Ouni, M.W. Mkaouer, R.G. Kuka, K. Inoue, WhoReview: a multi-objective search-based approach for reviewers recommendation in modern code review, Appl. Soft Comput. (2020) 106908.
[18] C. Yang, X.-h. Zhang, L.-b. Zeng, Q. Fan, T. Wang, Y. Yu, G. Yin, H.-m. Wang, Revrec: a two-layer reviewer recommendation algorithm in pull-based development model, J. Cent. South Univ. 25 (5) (2018) 1129–1143.

[19] M.B. Zanjani, H. Kagdi, C. Bird, Automatically recommending peer reviewers in modern code review, IEEE Trans. Softw. Eng. 42 (6) (2015) 530–543.
[20] J.B. Lee, A. Ihara, A. Monden, K.-I. Matsumoto, Patch reviewer recommendation in OSS projects, in: 2013 20th Asia-Pacific Software Engineering Conference, vol. 2, APSEC, IEEE, 2013, pp. 1–6.
[21] X. Xia, D. Lo, X. Wang, X. Yang, Who should review this change? Putting text and file location analyses together for more accurate recommendations, in: 2015 IEEE International Conference on Software Maintenance and Evolution, ICSME, IEEE, 2015, pp. 261–270.
[22] Y. Yu, H. Wang, G. Yin, T. Wang, Reviewer recommendation for pull-requests in GitHub: what can we learn from code review and bug assignment?, Inf. Softw. Technol. 74 (2016) 204–218.
[23] M. Fejzer, P. Przymus, K. Stencel, Profile based recommendation of code reviewers, J. Intell. Inf. Syst. 50 (3) (2018) 597–619.
[24] J. Kim, E. Lee, Understanding review expertise of developers: a reviewer recommendation approach based on latent Dirichlet allocation, Symmetry 10 (4) (2018) 114.
[25] M.L. de Lima Júnior, D.M. Soares, A. Plastino, L. Murta, Automatic assignment of integrators to pull requests: the importance of selecting appropriate attributes, J. Syst. Softw. 144 (2018) 181–196.
[26] J. Jiang, J.-H. He, X.-Y. Chen, Coredevrec: automatic core member recommendation for contribution evaluation, J. Comput. Sci. Technol. 30 (5) (2015) 998–1016.
[27] X. Ye, Learning to rank reviewers for pull requests, IEEE Access 7 (2019) 85382–85391.
[28] J. Jiang, Y. Yang, J. He, X. Blanc, L. Zhang, Who should comment on this pull request? analyzing attributes for more accurate commenter recommendation in pull-based development, Inf. Softw. Technol. 84 (2017) 48–62.
[29] H. Ying, L. Chen, T. Liang, J. Wu, Earec: leveraging expertise and authority for pull-request reviewer recommendation in GitHub, in: Proceedings of the 3rd International Workshop on CrowdSourcing in Software Engineering, ACM, 2016, pp. 29–35.
[30] Z. Liao, Z. Wu, Y. Li, Y. Zhang, X. Fan, J. Wu, Core-reviewer recommendation based on pull request topic model and collaborator social network, Soft Comput. (2019) 1–11.
[31] E. Sülün, E. Tüzün, U. Doğrusöz, Rstrace+: reviewer suggestion using software artifact traceability graphs, Inf. Softw. Technol. (2020) 106455.
[32] J. Lipcak, B. Rossi, A large-scale study on source code reviewer recommendation, in: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications, SEAA, IEEE, 2018, pp. 378–387.
[33] C. Hannebauer, M. Patalas, S. Stünkel, V. Gruhn, Automatically recommending code reviewers based on their expertise: an empirical comparison, in: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ACM, 2016, pp. 99–110.
[34] J. Jiang, D. Lo, J. Zheng, X. Xia, Y. Yang, L. Zhang, Who should make decision on this pull request? analyzing time-decaying relationships and file similarities for integrator prediction, J. Syst. Softw. 154 (2019) 196–210.
[35] S. Asthana, R. Kumar, R. Bhagwan, C. Bird, C. Bansal, C. Maddila, S. Mehta, B. Ashok, Whodo: automating reviewer suggestions at scale, in: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ACM, 2019, pp. 937–945.
[36] W.H.A. Al-Zubaidi, P. Thongtanunam, H.K. Dam, C. Tantithamthavorn, A. Ghose, Workload-aware reviewer recommendation using a multi-objective search-based approach, in: Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering, 2020, pp. 21–30.
[37] A. Strand, M. Gunnarson, R. Britto, M. Usman, Using a context-aware approach to recommend code reviewers: findings from an industrial case study, in: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice, 2020, pp. 1–10.
[38] S. Rebai, A. Amich, S. Molaei, M. Kessentini, R. Kazman, Multi-objective code reviewer recommendations: balancing expertise, availability and collaborations, Autom. Softw. Eng. 27 (3) (2020) 301–328.
[39] E. Mirsaeedi, P.C. Rigby, Mitigating turnover with code review recommendation: balancing expertise, workload, and knowledge distribution, in: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, 2020, pp. 1183–1195.
[40] A. Chueshev, J. Lawall, R. Bendraou, T. Ziadi, Expanding the number of reviewers in open-source projects by recommending appropriate developers, in: 2020 IEEE International Conference on Software Maintenance and Evolution, ICSME, IEEE, 2020, pp. 499–510.
[41] G. Gousios, The GHTorent dataset and tool suite, in: Proceedings of the 10th Working Conference on Mining Software Repositories, IEEE Press, 2013, pp. 233–236.
[42] O.E. Gundersen, S. Kjensmo, State of the art: reproducibility in artificial intelligence, in: Thirty-Second AAAI Conference on Artificial Intelligence, 2018.
[43] A. Bosu, J.C. Carver, C. Bird, J. Orbeck, C. Chockley, Process aspects and social dynamics of contemporary code review: insights from open source development and industrial practice at Microsoft, IEEE Trans. Softw. Eng. 43 (1) (2016) 56–75.
[44] T. R. Haynes, L. Sun, Code reviewer selection in a distributed software development environment, US Patent 9,595,009, Mar. 14 2017.
[45] M. Woulfe, Automatic identification of appropriate code reviewers using machine learning, US Patent App. 16/391,300, Oct. 29 2020.
[46] V. Balachandran, Automatic code review and code reviewer recommendation, US Patent 9,201,646, Dec. 1 2015.
[47] V. Kovalenko, N. Tintarev, E. Pasynkov, C. Bird, A. Bacchelli, Does reviewer recommendation help developers?, IEEE Trans. Softw. Eng. 46 (7) (2018) 710–731.
[48] K. Schwaber, J. Sutherland, The scrum guide - the definitive guide to scrum: the rules of the game, scrum.org, Nov-2017, 2017.
[49] J.M. González-Barahona, G. Robles, On the reproducibility of empirical software engineering studies based on data retrieved from development repositories, Empir. Softw. Eng. 17 (1–2) (2012) 75–89.
[50] D.M. Fernández, J.-H. Passoth, Empirical software engineering: from discipline to interdiscipline, J. Syst. Softw. 148 (2019) 170–179.
[51] D.M. Fernández, M. Monperrus, R. Feldt, T. Zimmermann, The open science initiative of the empirical software engineering journal, Empir. Softw. Eng. 24 (3) (2019) 1057–1060.
[52] E. Doğan, E. Tüzün, K.A. Tecimer, H.A. Güvenir, Investigating the validity of ground truth in code reviewer recommendation studies, in: 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM, IEEE, 2019, pp. 1–6.
[53] A. Ampatzoglou, S. Bibi, P. Avgeriou, M. Verbeek, A. Chatzigeorgiou, Identifying, categorizing and mitigating threats to validity in software engineering secondary studies, Inf. Softw. Technol. 106 (2019) 201–230.
[54] X. Zhou, Y. Jin, H. Zhang, S. Li, X. Huang, A map of threats to validity of systematic literature reviews in software engineering, in: 2016 23rd Asia-Pacific Software Engineering Conference, APSEC, IEEE, 2016, pp. 153–160.
[55] D. Badampudi, R. Britto, M. Unterkalmsteiner, Modern code reviews-preliminary results of a systematic mapping study, in: Proceedings of the Evaluation and Assessment on Software Engineering, ACM, 2019, pp. 340–345.