Code Reviewer Recommendation in Tencent: Practice, Challenge, and Direction*

Qiuyuan Chen*
Zhejiang University
Hangzhou, Zhejiang, China
chenqiuyuan@zju.edu.cn

Dezhen Kong*
Zhejiang University
Hangzhou, Zhejiang, China
timkong@zju.edu.cn

Lingfeng Bao†
Zhejiang University
Hangzhou, Zhejiang, China
lingfengbao@zju.edu.cn

Chenxing Sun
Tencent Technology
Shenzhen, Guangdong, China
marssun@tencent.com

Xin Xia
Zhejiang University
Hangzhou, Zhejiang, China
xin.xia@acm.org

Shanping Li
Zhejiang University
Hangzhou, Zhejiang, China
shan@zju.edu.cn

ABSTRACT

Code review is essential for assuring system quality in software engineering. Over decades in practice, code review has evolved to be a lightweight tool-based process focusing on code change: the smallest unit of the development cycle, and we refer to it as Modern Code Review (MCR). MCR involves code contributors committing code changes and code reviewers reviewing the assigned code changes. Such a reviewer assigning process is challenged by efficiently finding appropriate reviewers. Recent studies propose automated code reviewer recommendation (CRR) approaches to resolve such challenges. These approaches are often evaluated on open-source projects and obtain promising performance.

However, the code reviewer recommendation systems are not widely used on proprietary projects, and most current reviewer selecting practice is still manual or, at best, semi-manual. No previous work systematically evaluated these approaches' effectiveness and compared each other on proprietary projects in practice. In this paper, we performed a quantitative analysis of typical recommendation approaches on proprietary projects in Tencent. The results show an imperfect performance of these approaches on proprietary projects and reveal practical challenges like the "cold start problem". To better understand practical challenges, we interviewed practitioners about the expectations of applying reviewer recommendations to a production environment. The interview involves the current systems' limitations, expected application scenario, and information requirements. Finally, we discuss the implications and the direction of practical code reviewer recommendation tools.

CCS CONCEPTS

• **Software and its engineering** → Software maintenance tools.

KEYWORDS

code review, code reviewer recommendation, recommendation algorithm

ACM Reference Format:
Qiuyuan Chen*, Dezhen Kong*, Lingfeng Bao†, Chenxing Sun, Xin Xia, and Shanping Li. 2022. Code Reviewer Recommendation in Tencent: Practice, Challenge, and Direction. In *44nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '22), May 21–29, 2022, Pittsburgh, PA, USA*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3510457.3513035

1 INTRODUCTION

Code review (i.e., manual inspection of the source code) is an essential process that helps improve the software quality [3]. Early code review is a heavy process that focuses on the strict inspection of the source code like Fagan Style code inspection [7]. Over decades in practice, code review has evolved to be lightweight, focus on code changes, and adopts particular code review tools. This inspection process is referred to as Modern Code Review (MCR) in literature [19]. Even though code review has become a standard step in software development practice, MCR is still a labour-intensive process. To improve the efficiency, MCR aims to reduce time consuming and appears to be informal and asynchronous [19].

Modern code review involves two roles - code contributors and code reviewers. In a simplified code review procedure, code contributors commit code changes and propose a code review request; code reviewers respond to the request asynchronously, then review code changes and give feedback. However, practitioners often find reviewers manually or, at best, semi-manually in practice. Therefore, researchers proposed *Code Reviewer Recommendation (CRR)* (also known as code reviewer assignment or reviewer matching) approaches to recommend code reviewers efficiently [23, 28, 30–32]. The CRR approaches are expected to be embedded in the code review tools in which code contributors can invite code reviewers based on the list provided by the recommendation system.

These approaches are often achieved by learning the historical data of code review activity, including the developers' code

*Work done while this author was an intern at Tencent.

†Corresponding Author.

ICSE 2022, 15 - 19 May, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9226-6/22/05…$15.00
https://doi.org/10.1145/3510457.3513035

changes and participation in prior code reviewers. Previous works often evaluate these approaches on open-source projects, and most approaches obtain promising performance, sometimes as high as 92% [12] in terms of accuracy@top-5, one of commonly used metrics to evaluate a code reviewer recommendation approach. However, most current reviewer selecting practice is still manual or, at best, semi-manual. For example, a popular code review tool, Gerrit, can only invite code reviewers by typing names. There are attempts to attack the challenge of building a CRR system in practice [2, 18, 21], but little previous work systematically evaluated the effectiveness of existing approaches and compare each other on proprietary projects in practice.

Tencent is an international internet and technology company with more than 90,000 employees. It has a range of business areas such as communication and social, cloud computing, advertising, FinTech, and other enterprise services, containing tens of thousands of code projects in different areas. Nowadays, Tencent is promoting inner-source practice, which aims to improve the coordination of proprietary projects. Inner-source projects require the proprietary projects to be readable and useful for other developers in the company. With hundreds of thousands of projects, up to 70% of projects in Tencent are inner-sourced. However, inner-source makes it harder to find appropriate code reviewers as developers may not be as familiar with each other as in the same project. The code review tool in Tencent matches code reviewers based on manually maintained configuration files, suffering from a lack of scalability and flexibility. At the same time, existing machine-learning-based CRR approaches can update and recommend automatically. Therefore, we investigate the first research question:

RQ1: What is the effectiveness of code reviewer recommendation approaches on proprietary projects?

We select top ten proprietary projects with the largest number of reviewers, covering areas of infrastructure, Fintech, database, and so on. We select five typical existing code reviewer recommendation algorithms, i.e., RevFinder [22], TIE [28], IR (VSM-based) [31], Comment Network (CN) [31], and cHRev [32]. Some other approaches are excluded because of the data limitation in practice (e.g., Carrot [21] requires context information; WLR-REC [1] requires confidential personal information).

We apply these approaches to the selected projects and found they do not perform as well as the original reports. We find that project characteristics can impact the effectiveness of CRR approaches, and projects with "dominant reviewers" are intended to perform well. However, CRR approaches can be less helpful in this kind of project and even skews the reviews assignment. In particular, we notice a "cold start problem" that cause an ML-based CCR recommendation system to be invalid when it is applied to a new project.

Based on the findings in the quantitative analysis, we further conduct a qualitative analysis to understand the practical challenges. We try to answer the second research question:

RQ2: What are the perceptions and expectations of practitioners on the reviewer recommendation?

We interviewed 11 (two for pilot interview and nine for formal interview) practitioners about the reviewer recommendation systems. The interviewees discussed the limitation of the current reviewer recommendation system, expected application scenario, and information requirements.

We find that current configuration-based recommendations in Tencent can support daily requirements of finding reviewers when contributor-reviewer relationships are relatively stable. But it suffers from scalability and environmental change like staff turnover [17]. In addition, interviewees mentioned that inviting too many reviewers does cause a "notification noise" issue, which requires a trade-off between the recommendation size and the accuracy. For the expectation on application scenarios, practitioners are optimistic about the machine-learning-based CRR approaches. However, we should consider various situations (e.g., targets passing code review quickly or improving top-level design by design) to design a practical CRR system. Last, practitioners expect the CRR system to consider as much code review related information as possible for the information requirements. Apart from improving algorithm accuracy, we also suggest the CRR system improve information transparency to help code contributors and code reviewers get familiar with each other, facilitating the reviewer selecting process and improve user experience.

In summary, our study makes the following contributions:

- We apply various existing CRR approaches and compare the performance differences on proprietary projects in Tencent. We evaluate the approaches from different dimensions, including tool usability and scenario feasibility.
- We interview 11 professionals from different areas to shed light on practitioners' perceptions and expectations towards the CRR system in practice.
- We present the results of quantitative and qualitative of CRR in practice. We highlight the challenges of applying the existing approaches in practice and provide suggestions to meet practitioners' expectations better.

We believe that our findings can help improve CRR system from the perspectives of the algorithm, developer perception, and scenario in practice. The remainder of the paper is organized as follows: Section 2 describes the background and related work. Section 3 shows the methodology of this paper, including the experiment methodology for RQ1 and the interview methodology for RQ2. Section 4 shows the results of the two research questions and illustrates our findings. Section 5 discusses the implications and the influence of code reviewer recommendation on the code review process. Section 6 describes the threats to the validity of this paper. Section 7 concludes the paper and shows our future work.

2 BACKGROUND AND RELATED WORK

2.1 Code Review Workflow in Tencent

Tencent has many projects in various business areas leading to customized code review practice in detail. We briefly describe a common code review workflow and highlight the **code reviewer recommendation scenario** as follows.

- (S1) At the beginning of a code review workflow, a contributor fetches code changes and contribute their own code to the repositories. This process will ask to create a code review request if code review on this branch is configured to be required.

- **(S2) When creating the CR request, the system can give a list of recommended reviewers for the code contributors to select. The recommendation is based configuration files recording ownership and authorship of code. Contributors can also added reviewers outside of the recommended list by searching qualified reviewers. Then the final list will be sent out to invite candidate reviewers.**
- (S3) The candidate reviewers can choose to accept, reject, or ignore the request. Reviewers who accept the request will finish code review and give feedback on the code changes(*"accept"*, *"reject"*, or *"need revision"*). Contributors can further revise their commits to address the feedback. The selected reviewers stay the same for the next iteration.
- (S4) The contributed code changes can be merged if they pass the code review, or they will be abandoned if they are evaluated not suitable and should be given up.

Code reviewer recommendation works in S2, which is the research target in the scenarios mentioned above because it is embedded in the code review process. Some tools may provide separate tools for recommendation [21], but they may distract developers, causing more development costs.

2.2 Code Reviewer Recommendation in Practice

We investigate the code reviewer recommendation embedded in code review tools. Many open source organizations conduct code reviews on Gerrit[1], Reviewboard[2], or Git-based platforms (e.g. GitHub). Giant companies also develop code review products (e.g., Phabricator[3] by Facebook, and Upsource[4] by JetBrain), and they often adopt more than one code review tool for different needs. For example, Google adopts several tools, including their tool Critiques [19]; Microsoft also adopts different internal code review tools, including their CodeFlow [20] and in-project customized tools [12]. These tools provide basic support of code reviewer recommendations. For example, Critique recommends code reviewers based on the reviewed history on particular files [19].

There are attempts to attack the challenge of building a CRR system in practice [2, 10, 12, 15, 21]. Strand et al. [21] deployed a code reviewer recommendation system called Carrot in Erricson. Carrots adopt the LightFM algorithm, which combines collaborative filtering and context-based filtering and trains the model on the code changes in the history of the CR system (i.e., Gerrit). They performed interviews and user studies to confirm the effectiveness. Asthana et al. [2] propose WhoDo, which recommends reviewers based on (1) whether a developer reviewed/committed a particular file/directory and (2) load balancing of a developer. They deploy the system on five repositories within Microsoft and analyze the results in practice qualitatively and quantitatively. Kovalenko et al. [12] deployed an IR-based CRR system in two companies (Microsoft and JetBrain) and analyzed practical data in production environments. They also interviewed and surveyed practical users to measure the influence of recommendations on users' choices. Unlike their work, our study analyzes the effectiveness of state-of-the-art existing CRR algorithms and compare each other instead of analyzing CRR based on a deployed tool. We also asked the developers about their expectations and suggestions to help design a better practical tool. Kagdi et al. [11] propose an approach to recommend reviewers by estimating how likely one has good knowledge of the target PRs. It utilizes several heuristics to measure candidates' expertise, change activity, and commit contributions. Ying et al. [29] propose an approach to recommend reviewers by constructing a graph architecture to depict the expertise and authority of developers as well as their interactions. In this work, we perform an empirical study evaluating various existing CRR approaches using proprietary projects and summarizing practical implications by interview.

Research of code reviewer recommendation involves abundant technical and empirical studies. Due to the space limitation, this paper only analyses CRR in practical tools and introduces several works related to the production environment. We convey more related CRR approaches to a comprehensive survey. Please refer to the review studies proposed by Cetin et al. [33].

3 RESEARCH METHODOLOGY

3.1 Overview

Figure 1 shows an overview of our research methodology which consists of two stages.

Stage 1: We performed experiments on the top ten proprietary projects that contained the largest number of reviewers. The experimental results provide the implications of current code reviewer recommendation approaches; it also motivates us to interview practitioners in the next stage. The experimental results also help us design the interview guide.

Stage 2: We interview professionals on their perceptions of CRR approaches. All interviewees come from the projects in stage 1. In this stage, we have a pilot interview with professionals of Tencent's current code review tool and designed an interview guide. Then we conduct a formal semi-structured interview with practitioners recruited from different areas in Tencent.

3.2 Experiment Methodology

3.2.1 Project Selection and Data Characteristics. Intuitively, the developer number in a project can indicate whether a project needs to adopt a code reviewer recommendation system. For example, a project with only several developers can find code reviewer easily. In contrast, a large size of a project can make it hard to find appropriate code reviewers. In this paper, we select ten projects that contains the largest number of reviewers in Tencent for experimental analysis. We extract all historical developing and reviewing records in these projects. The analysis granularity is "one" code review in which all iterations (in cases of "need revision") will be attached to this one. Then we exclude the noise and invalid information in these data (e.g., automatically generated commits). Finally we perform different code reviewer recommendation algorithms on these projects.

3.2.2 CRR Approach Selection. There are many code reviewer recommendation approaches (CRR approaches) in literature [28, 31]

[1]https://www.gerritcodereview.com/
[2]https://www.reviewboard.org/
[3]https://secure.phabricator.com/
[4]https://www.jetbrains.com/upsource/

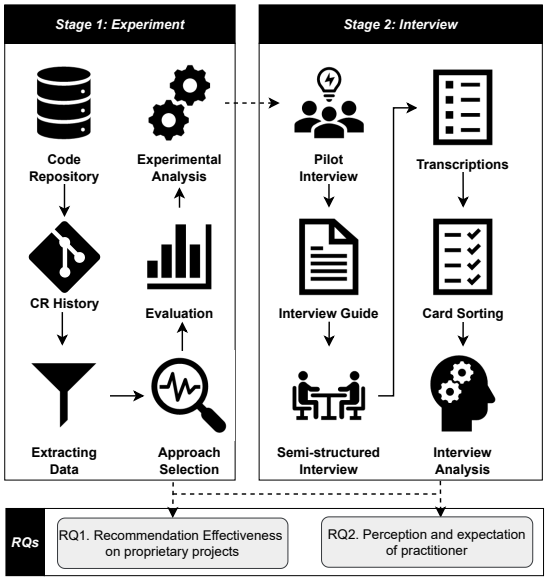

Figure 1: Research methodology overview

considering different factoring during the code review process. We focus on the historical data in the projects including commits and code reviews. We choose the following representative traditional recommendation approaches, which are still state-of-the-art ones according to the study performed by Hu et al. [9]. The selected code reviewer recommendation approaches are listed as follows:

(1) **RevFinder** [23] is an expertise-based approach that leverages file paths, assuming that the files located in close files may share similar functionality and are likely to be reviewed by reviewers with common experience.
(2) **TIE** [28] uses multinomial Naive Bayes to measure the commit message's textual content (i.e., commit message) similarity and a VSM-based approach to measure the file path similarity. Then it combines the weighted scores and ranks them to recommend the reviewers.
(3) **IR (VSM-based)** [31] *vectorizes* the PR's description using VSM, calculates the textual similarities, and ranks the reviewers in the resolved PRs.
(4) **Comment Network (CN)** [31] is a recommender that ranks reviewers who share common interests with the contributors of target PR by mining historical comments traces and construct a comment network. Intuitively, developers who always review the author's code change or the dominant participants of the project are more tended to be recommended.
(5) **cHRev** [32] considers the reviewing history (review number, review time). It counts the number of comments to the file as part of scores. It also considers the frequency and recency of historical contributors to measure the reviewer expertise and give a final recommendation.

Besides, We do not include some approaches because of the data availability (e.g., libraries specific to the current code changes) and sensitivity (e.g., personal information like records of developers' time and expertise experience). For example, Carrot [21] requires context information and WLR-REC [1] requires personal information.

3.2.3 Metrics. We adopt commonly used metrics *top-k accuracy MRR*, precision, and recall to evaluate CRR approaches. The first two metrics are widely used in literature of software engineering [23, 28]. We denote a reviewer as r and multiple reviewers as Rs; denote a code review as cr and multiple code reviews as CRs.

Top-k Accuracy: *Top-k accuracy* is the ratio of code reviews where their ground truth reviewer are ranked within the top-k positions in the recommended ranked lists of reviewers. Specifically, given a code review cr, we can recommend a ranked list of reviewers Rs. If at least one of its top-k code reviewers actually reviews the code review cr (i.e., *Reviewed(cr, top-k)*), we consider the reviewers are correctly recommend, and set the value *Reviewed(cr, top-k)* to 1; otherwise if there is no one actually reviews the cr, we set the value to 0, which means it is wrongly recommended. Given a set of code reviews CRs (chronological order), the top-k prediction accuracy is computed as:

$$Top@k = \frac{\sum_{cr \in Reviewed(cr,top-k)}}{|CRs|} \quad (1)$$

The higher the value, the better a reviewer recommendation approach is. In Practice, *top-k accuracy* measure the probabilistic of hitting at least one valid reviewer so that the code change can be reviewed. In literature, we set k=1, 3, 5, and 10, which is the same as previous studies [9, 28].

Mean Reciprocal Rank (MRR): *MRR* is a popular metric for information retrieval technique [4] that is suitable for measuring the ranked recommendation list. Given a code-review cr, its reciprocal rank is the inverse of the first correct reviewer r in a rank list of the recommendation. Mean Reciprocal Rank (*MRR*) is the average of the average of the reciprocal ranks of code-reviews in a set of chronological code-reviews. The *MRR* of a set of code-reviews CR is computed as:

$$MRR(R) = \frac{1}{|Reviews|} \sum_{r \in Reviews} \frac{1}{rank(r)} \quad (2)$$

where $|CRs|$ denotes the number of code-reviews and *rank(r)* refers to the rank of the first reviewer r in the recommended list. The higher the *MRR*, the higher the ranking of the first correctly recommended reviewer is.

Precision and recall are also common metrics used to evaluate the effectiveness of CRR in previous work and we adopt the same definitions [32].

3.3 Interview Methodology

3.3.1 Protocol. The first author of this paper conducted a series of face-to-face interviews with nine software practitioners with experience of reviewing code and being reviewed by others[5]. Each interview took 35-45 minutes. We observed saturation of opinions when our interviews were near the end.

Pilot Interview. The interview was semi-structured, which is based on *an interview guide*. The guide contains general groupings of topics and questions instead of a pre-determined specific

[5]Participants were instructed that we wanted their opinions; privacy and sensitive resources were not explicitly mentioned.

set and order of questions. We build the *interview guide* in the pilot interview with two interviewees. The pilot interview is an open-ended discussion with senior developers developing or maintaining the code review tool in the company. Interviewees in the pilot interview are more familiar with the reviewer recommendation than other developers. They introduce the hands-on experience of finding code reviewers with assistance with a recommendation system. The pilot interview helps us cover more topics and improve the expression of questions. During the pilot interview and literature review, we iteratively refine the questions in the *interview guide*, which is used in the formal interview.

The formal interview consists of four parts in Table 1. Table 1 summarizes the main content and complete *interview guide* is online[6]. In the first part, we asked demographic questions about the programming preference, especially the interviewee's experience reviewing code and being reviewed by others. We also provided interviewees with the experimental results in RQ1, explained the detailed metrics, and asked their opinions about the performance.

In the second part, we asked several open-ended questions about how they find appropriate code reviewers in assistance with a recommendation system and the perceptions of the current system. The purpose of this part is to allow the interviewees to express feelings and expectations freely about the practical scenarios of finding appropriate code reviewers.

In the third part, we presented interviewees with three lists of topics covering three aspects (i.e., detailed feedback, scenarios, and algorithm). We asked the interviewees to discuss topics that they had not explicitly mentioned. The first list comes from the pilot interviews to further investigate the detailed feedback of the current recommendation system. The second list is designed to discuss the scenarios of the reviewer recommendation system. The third list briefly introduced the algorithm of the machine-learning-based recommender. We asked interviewees to discuss the potential information that we could use to augment the algorithm. The third list also combines the questions in literature [12]. We choose the three aspects (detailed feedback, scenarios, and algorithm) to ensure that we can fully cover the topics of (1) understanding the practitioners' requirements, (2) expected application scenarios, and (3) selecting practical and cost-effective algorithms with necessary information.

In the fourth part, we presented rating questions (in 5-point Likert scale: from Strongly Disagree to StronglyAgree) about the relevant topics of code reviewer recommendation, including knowledge sharing, environment challenges, tool feasibility, and reviewing workload. The purpose of this part is to investigate how the reviewer recommendation can impact other code review activities.

At the end of both the pilot and formal interviews, we allowed interviewees to provide free-text comments, suggestions, and opinions about code reviewer recommendations and our interview. An interviewee may or may not provide any final comments. Last, we thanked the interviewees and briefly introduced our next plan.

3.3.2 Participant Selection. We recruited full-time employees familiar with modern code review (MCR) in Tencent. This study is intended to improve the code review tool experience so we can communicate with the developers with on-hand experience about reviewer recommendation in the pilot interview as mentioned in Section 3.3.1. We recruited interviewees by contacting the active users of the in-company code review tool, who have a strong willingness to provide suggestions to improve the code review process.

In this way, we recruited 11 interviewees (two for aforementioned pilot interview and nine for formal interview), and they come from various business areas. All interviewees have an average of 7.1 years of professional experience (min: 4, max: 11, median: 5). Most interviewees preferred to describe themselves as multi-language users, including Java, C++, Objective-C, Python, and other script languages. Interviewees can only give a rough number of reviewing/being during a particular period because it is not constant. Most interviewees said performing 3-4 code reviews on each day is suitable for them. All interviewees describe their role as the developer, and two interviewees also take responsibility for project management. In the remainder of the paper, we denote these interviewees as I1 to I9.

Table 1: Summarization of four parts in the interview guide.

Part I: Demographic
Part II: Open-ended Discussion
Discussion 2.1: feelings and perceptions
Discussion 2.2: user experience improvements
Part III: Specific Topic Discussion
Discussion 3.1: Existing Practice Feedback
Topic 1: can current CRR system meets need Topic 2: find reviewers in unfamiliar scenario Topic 3: deal with inappropriate reviewers Topic 4: deal with wrongly assigned reviewers Topic 5: Information for selecting reviewers
Discussion 3.2: Code Review Recommendation Scenario
Topic 1: code review scenario Topic 2: inner-source code review experience Topic 3: differences between inner-source and open-source
Discussion 3.3: Code Review Recommendation Algorithm
Topic 1: expected algorithm Topic 2: "hidden information" requests Topic 3: algorithm improvements
Part 4: Statement Agreements

3.3.3 Interview Data Analysis. To analyze the recorded interview data, we conduct a thematic analysis [6] coupled with open card sorting. We adopt the following steps after completing the last interview.

Transcribing and Coding. We transcribed and verified the recordings of the interviews and built a comprehensive understanding by reviewing the transcripts. The first author of this paper read the transcripts and coded the interviews using NVivo qualitative analysis software [27] in which similar statements or meanings are marked as the same code number. The second author then verified the initial codes created by the first author and helped improve the way of coding. Last we merged the codes with nearly identical or meanings and extract unique codes. We noticed that when the

[6]Interview guide online: https://mfr.osf.io/render?url=https%3A%2F%2Fosf.io%2Fvcqpe%2Fdownload

Table 2: Counts of the selected proprietary projects.

Project ID	Time Period	# Reviews	# Reviewers	# Files	# Review Per Day
P1	2018/10 - 2020/07	9,273	84	25,845	15.05
P2	2018/11 - 2021/06	24,413	195	33,717	25.86
P3	2019/02 - 2021/04	159	45	1,596	0.2
P4	2019/03 - 2021/06	674	87	2,143	0.81
P5	2019/09 - 2021/06	395	54	1,341	0.63
P6	2019/10 - 2021/06	1,028	165	177	1.74
P7	2019/11 - 2021/06	1,970	118	3,215	3.46
P8	2019/12 - 2020/09	177	49	262	0.63
P9	2020/01 - 2021/06	242	66	863	0.47
P10	2020/02 - 2021/06	3,060	77	3,556	6.18

interviews were near the end, the collected codes reached saturation. There are no new codes that emerged, and the code list was considered stable.

Open Card Sorting. This paper's first two authors (labellers) read and analyzed the derived codes separately and rephrased them into cards. Then they sorted the generated cards and turned them into potential themes for thematic similarity (used by Latoza et al. [14] and Wan et al. [24]). The themes are not pre-defined before the sorting phase. There can be bias caused by different subjective cognition, so we then use the Cohen's Kappa measure [8] to examine the agreement between the two labellers. The overall Kappa value between the two labelers is 0.76, which indicates substantial[7] After completing the labelling process, the two labellers discussed the disagreements to reach a consensus. To further reduce the bias from the two authors, they both reviewed and agreed on the final set of themes. Last, we derived the final statements that describe the status and expectations of the code reviewer recommendation in practice.

4 RESULTS

In this section, we show the results of two research questions. For the first research question, we illustrate the results on proprietary projects and further investigate the factors that influence the effectiveness. For the second research question, we organize the interviewees' opinions in the four parts separately and summarize the practical findings in the interview.

4.1 RQ1: Effectiveness of CRR Approaches on Proprietary Projects in Tencent

Table 2 shows the characteristics of the selected ten projects. Table 3 shows the experimental results of the selected approaches on projects from P1 to P10, covering areas of infrastructure, Fintech, database, and so on. We refer to such an analysis as a *retrospective analysis* as it is performed based on the historical data.

We can observe the results from the perspective of different metrics in Table 3. As the results are very informative, we analyze from different perspectives to derive findings and discuss implications. In a typical recommendation scenario, MRR can show the rank of the correctly recommended reviewers when a particular algorithm or scores order the results. Here it has a slightly different meaning because the current system does not show reviewer lists sorted in particular. Instead, it records the original order of selecting candidate reviewers. In other words, contributors invite code reviewers by adding them into the candidate list one by one, and the system can record such a process. Therefore, MRR can reflect how perfectly the reviewer recommendation approaches can fit this "selection process". We can observe that MRR scores range from 0.2 (IR) to 0.47 (CN), which is equivalent to average ranks range from 5 to 2.1 in the recommendation list. In other words, on average, the approaches present an actual reviewer in ranks from 5 (IR) to about 2 (CN) of the list of recommendations. We find the accuracy of different recommendation numbers has the same descending trend (i.e., the larger the number, the lower the performance). In particular, we can pay attention to the top-5 accuracy as it reflects a practical usage scenario: when a contributor launches a code review request, the system showed a recommendation list where the number is about 5 (also mentioned in Section 2). The best-performed approach, CN, reaches a score of 0.64, which means in about three in five (we can also treat it as a probabilistic) reviews, at least one reviewer could be actually recommended. In contrast, other approaches cannot reach even a half. Finally, precision and recall have nearly the same trend as accuracy, except that precision decreases when the recommendation number enlarges. Most reviews require only one code reviewer, thus the more it recommends, the lower the precision is.

We find that the selected approaches do not perform as well in the original reports [23, 28, 30–32], which is applied on open-source projects[8]. Even though ten projects are hard to cover all situations and a larger scale may draw different conclusions, these projects can still be representative of massive proprietary projects with many developers.

> **Finding 1.** The selected CRR approaches perform worse on proprietary projects than open-source projects.

From the perspective of approach differences, CN performs the best on nearly all metrics, considering the average results. In contrast, IR has the worst performance. As for the project perspective, average results cover up the differences between the projects; we can see that CRR approaches in P8 and P9 perform better in nearly all metrics. In particular, CN reaches 0.93 of accuracy@top-5 on P8, which is considered to be very promising. Therefore, we further investigate how the project characteristics can impact the results. By analyzing the reviewer distributions on code reviews, we notice a phenomenon that several reviewers covered a large proportion of code reviews. For example, in P8 and P10, the top three reviewers participated in over a half of total code reviews even when these their projects were large (involving 49 and 77 reviewers as in Table 2). We describe a project with "dominant reviewers" when less than five reviewers participate in more than a half code reviews. The project should be middle or large (more than 20 developers) as small projects naturally have "dominant reviewers". Note that our definition is based on ten proprietary projects in Tencent, and a more accurate definition and precise thresholds could be given by a larger scale study. When there are "dominant reviewer" in projects, CRR approaches that only involves the experience of commit histories may achieve a relatively lower performance (i.e., approaches except for CN), which is consistent with the explanation by Yu et al. [31]. Besides, in projects with "dominant reviewers", CRR

[7] Kappa value of [0.01, 0.20], (0.20, 0.40], (0.40, 0.60], (0.60, 0.80], and (0.80, 1] is considered as slight, fair, moderate, substantial, and almost perfect agreement, respectively

[8] We also performed replication experiments to confirm the conclusions as in Section 5

Table 3: MRR and accuracy, precision, and recall of top 1, 3, 5, 10 of the selected approaches on ten proprietary projects.

Approach	Project	MRR	top1@acc.	top3@acc.	top5@acc.	top10@acc.	top1@prec.	top3@prec.	top5@prec.	top10@prec.	top1@recall	top3@recall	top5@recall	top10@recall
RevFinder	P1	0.16	0.06	0.19	0.30	0.46	0.06	0.06	0.06	0.05	0.06	0.19	0.29	0.45
	P2	0.27	0.14	0.31	0.46	0.60	0.14	0.10	0.09	0.06	0.14	0.31	0.45	0.59
	P3	0.07	0.00	0.17	0.17	0.17	0.00	0.06	0.03	0.02	0.00	0.17	0.17	0.17
	P4	0.17	0.06	0.23	0.31	0.52	0.06	0.08	0.06	0.05	0.06	0.23	0.31	0.52
	P5	0.15	0.13	0.16	0.17	0.18	0.13	0.05	0.03	0.02	0.13	0.16	0.17	0.18
	P6	0.13	0.10	0.16	0.17	0.24	0.10	0.05	0.03	0.02	0.10	0.16	0.17	0.23
	P7	0.20	0.13	0.21	0.29	0.45	0.13	0.07	0.06	0.05	0.10	0.19	0.26	0.41
	P8	0.60	0.33	0.89	0.89	0.93	0.33	0.31	0.20	0.11	0.23	0.72	0.75	0.78
	P9	0.42	0.27	0.51	0.73	0.73	0.27	0.19	0.17	0.09	0.19	0.37	0.59	0.63
	P10	0.50	0.33	0.64	0.73	0.79	0.33	0.24	0.18	0.10	0.21	0.48	0.59	0.67
	Average	**0.27**	**0.16**	**0.35**	**0.42**	**0.51**	**0.16**	**0.12**	**0.09**	**0.06**	**0.12**	**0.30**	**0.38**	**0.46**
TIE	P1	0.37	0.24	0.36	0.53	0.67	0.24	0.12	0.11	0.07	0.21	0.33	0.49	0.63
	P2	0.24	0.11	0.27	0.37	0.57	0.11	0.09	0.07	0.06	0.09	0.21	0.28	0.45
	P3	0.06	0.02	0.04	0.06	0.15	0.02	0.01	0.01	0.01	0.02	0.04	0.06	0.15
	P4	0.16	0.07	0.15	0.22	0.41	0.07	0.05	0.04	0.04	0.07	0.15	0.22	0.41
	P5	0.35	0.20	0.44	0.53	0.60	0.20	0.15	0.11	0.06	0.20	0.44	0.52	0.59
	P6	0.19	0.11	0.23	0.28	0.33	0.11	0.08	0.06	0.03	0.11	0.23	0.28	0.33
	P7	0.21	0.14	0.20	0.26	0.37	0.14	0.07	0.05	0.04	0.12	0.18	0.24	0.35
	P8	0.51	0.28	0.76	0.76	0.80	0.28	0.25	0.16	0.09	0.18	0.47	0.50	0.55
	P9	0.44	0.24	0.55	0.70	0.80	0.24	0.20	0.17	0.10	0.17	0.37	0.52	0.61
	P10	0.46	0.22	0.67	0.74	0.80	0.22	0.24	0.16	0.09	0.17	0.56	0.63	0.69
	Average	**0.30**	**0.16**	**0.37**	**0.45**	**0.55**	**0.16**	**0.13**	**0.09**	**0.06**	**0.13**	**0.30**	**0.37**	**0.48**
IR	P1	0.25	0.07	0.33	0.52	0.71	0.07	0.11	0.10	0.07	0.05	0.28	0.47	0.66
	P2	0.17	0.04	0.18	0.37	0.60	0.04	0.06	0.07	0.06	0.03	0.15	0.29	0.49
	P3	0.02	0.00	0.06	0.06	0.06	0.00	0.02	0.01	0.01	0.00	0.06	0.06	0.06
	P4	0.05	0.00	0.04	0.06	0.30	0.00	0.01	0.01	0.03	0.00	0.04	0.06	0.30
	P5	0.07	0.03	0.09	0.11	0.16	0.03	0.03	0.02	0.02	0.03	0.09	0.11	0.16
	P6	0.08	0.05	0.09	0.13	0.17	0.05	0.03	0.03	0.02	0.05	0.09	0.13	0.17
	P7	0.19	0.13	0.21	0.27	0.39	0.13	0.07	0.05	0.04	0.10	0.18	0.14	0.36
	P8	0.45	0.31	0.44	0.74	0.81	0.31	0.18	0.18	0.11	0.20	0.33	0.54	0.63
	P9	0.20	0.11	0.21	0.31	0.52	0.11	0.08	0.06	0.06	0.07	0.16	0.22	0.37
	P10	0.51	0.36	0.61	0.67	0.80	0.36	0.23	0.16	0.10	0.22	0.44	0.49	0.65
	Average	**0.20**	**0.11**	**0.23**	**0.32**	**0.45**	**0.11**	**0.08**	**0.07**	**0.05**	**0.08**	**0.18**	**0.25**	**0.39**
CN	P1	0.41	0.24	0.51	0.64	0.85	0.24	0.17	0.13	0.09	0.24	0.50	0.63	0.84
	P2	0.67	0.57	0.77	0.83	0.86	0.57	0.26	0.17	0.09	0.56	0.75	0.81	0.85
	P3	0.26	0.20	0.30	0.30	0.50	0.20	0.10	0.06	0.05	0.20	0.30	0.30	0.50
	P4	0.50	0.41	0.57	0.63	0.70	0.41	0.19	0.13	0.07	0.40	0.57	0.63	0.70
	P5	0.58	0.51	0.66	0.70	0.71	0.51	0.22	0.14	0.07	0.50	0.64	0.68	0.70
	P6	0.28	0.21	0.32	0.40	0.47	0.21	0.11	0.08	0.05	0.21	0.32	0.40	0.47
	P7	0.42	0.24	0.57	0.68	0.75	0.24	0.19	0.14	0.07	0.24	0.52	0.63	0.70
	P8	0.60	0.33	0.89	0.93	0.93	0.33	0.32	0.21	0.12	0.23	0.73	0.78	0.83
	P9	0.48	0.33	0.56	0.67	0.80	0.33	0.21	0.17	0.10	0.23	0.42	0.57	0.69
	P10	0.50	0.35	0.62	0.66	0.78	0.35	0.23	0.15	0.10	0.22	0.45	0.49	0.64
	Average	**0.47**	**0.34**	**0.58**	**0.64**	**0.74**	**0.34**	**0.20**	**0.14**	**0.08**	**0.30**	**0.52**	**0.59**	**0.69**
cHRev	P1	0.24	0.16	0.28	0.35	0.47	0.16	0.09	0.07	0.05	0.15	0.27	0.34	0.46
	P2	0.32	0.23	0.35	0.45	0.55	0.23	0.12	0.09	0.06	0.22	0.34	0.44	0.54
	P3	0.04	0.00	0.00	0.00	0.28	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.28
	P4	0.27	0.20	0.31	0.37	0.44	0.20	0.10	0.07	0.04	0.20	0.31	0.37	0.44
	P5	0.49	0.39	0.56	0.63	0.68	0.39	0.19	0.13	0.07	0.38	0.55	0.62	0.67
	P6	0.18	0.12	0.23	0.27	0.30	0.12	0.08	0.05	0.03	0.12	0.23	0.27	0.30
	P7	0.33	0.23	0.41	0.47	0.51	0.23	0.14	0.09	0.05	0.20	0.38	0.43	0.47
	P8	0.11	0.04	0.07	0.07	0.39	0.04	0.02	0.01	0.04	0.02	0.04	0.04	0.27
	P9	0.27	0.17	0.31	0.41	0.49	0.17	0.12	0.10	0.06	0.11	0.24	0.32	0.39
	P10	0.64	0.49	0.74	0.84	0.89	0.49	0.30	0.21	0.12	0.34	0.60	0.71	0.80
	Average	**0.29**	**0.20**	**0.33**	**0.39**	**0.50**	**0.20**	**0.12**	**0.08**	**0.06**	**0.17**	**0.30**	**0.35**	**0.46**

approaches are intended to recommend "dominant reviewers" repeatedly. CRR approaches in such projects are likely to be less helpful as practitioners may not need such recommendations, and the repetition skewed reviews assignment in turn.

> **Finding 2.** We find that project characteristics can impact the effectiveness of CRR approaches, and projects with "dominant reviewers" are intended to perform well.

We notice a "cold start problem" when discussing how to embed the code reviewer recommendation in a practical code review tool. "Cold start problem" is a common issue in recommendation system [13, 26]. It happens when machine-learning-based CRR approaches do not learn a good-enough model at the beginning of a project, determining the feasibility of a newly deployed CRR approach in turn.

Figure 2 shows the performance (average top5@accuracy and MRR of the ten projects) of the CN (the best performed CRR approach) in chronological order. We get the results by extracting the performance of CN week by week. We can observe that their performance of CN suffers from a relatively low performance at the first several weeks and fluctuations before reaching the best performance. Similarly, other selected CRR approaches suffer from the "cold start problem" that the performance cannot increase in a short time at the beginning [5]. In particular, on the project P9, all CRR approaches fail to hit any correct reviewers for a long time (i.e., get zero scores for up to three weeks). We describe it as a project-level "cold start problem" in CRR.

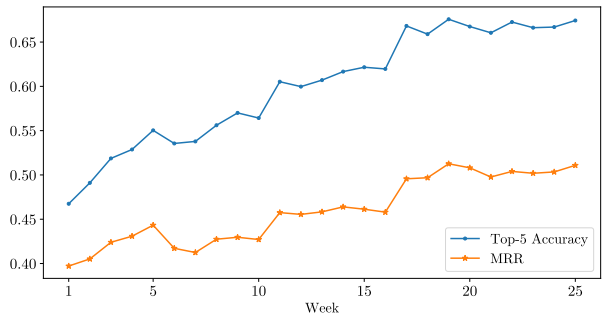

Figure 2: Average top-5 accuracy and MRR of CN (the best performed CRR approach) on ten proprietary projects in chronological order (by weeks).

Though the project-level "cold start problem" is not explicitly touched before, a prior study proposed by Asthana et al. mentioned a different "cold start problem" in CRR that [2] when new files are added to the repositories, they are not possible to directly link the new files to the potential reviewers. We describe it as a file-level "cold start problem". We can treat the file-level one as a partial explanation of the project-level "cold start problem" because CRR approaches need time to learn the weight of new files (if the approach involves files). Many machine-learning-based CRR approaches train models chronologically, making it hard to avoid "cold start" when there are little data at the beginning. We can consider algorithm improvement (e.g., few-shot learning [25] for an early phase) or adopt other strategies (e.g., manual rules) to alleviate the "cold start problem".

Finding 3. "Cold start problem" impacts the machine-learning-based CRR approaches, determining the feasibility and effectiveness of newly deployed approaches.

4.2 RQ2: Interview with Practitioners on Code Reviewer Recommendation

The formal interview consists of four parts, as mentioned in Section 3.3.1. We show the summarized results and the findings for each part, respectively.

The second part is an open-ended discussion (the **Part II** as in Table 1). We asked interviewees about the feedback of using the current reviewer recommendation system (i.e., the scenario S2 mentioned in background Section 2). The current system is configuration-based which records the authorship and ownership of code changes. More than a half of interviewees (5) said the current system could satisfy their requirements in daily development because when the contributor-reviewer relationships are relatively stable, such a fixed configuration can support the requirements of finding reviewers.

However, such a configuration is maintained manually, which cannot update in time if the contributor-reviewer relationship changes. We categorize such "relationship change" into four kinds of situations mentioned in the interview. First, staff turnovers can make the configuration invalid. For example, I6 said *"Sometimes such a configuration is the only resource for me to find code reviewers. However, it can be invalid when the recorded developer resigns."*. It is also noticed in previous work [17]. Second, newcomers who are not familiar with the project cannot be assigned to the configuration, leading to delayed updates. Third, the configuration is invalid for some inner-source projects as it is hard to maintain such a configuration. For example, I8 said in some inner-source projects, *"I can only ask the project leader to manually identify who should review, causing a time waste"*. Last, some legacy system lacks such a code ownership configuration. In summary, the configuration-based system cannot assure scalability, and the configuration accuracy decays quickly, confronting the situations above.

Finding 4. When the contributor-reviewer relationship is relatively stable, configuration-based recommendations support daily requirements of finding reviewers. However, the manual-maintained configuration cannot assure scalability, and its quality decays quickly.

For the third part, we asked several aspects of code reviewer recommendation (the **Part III** shown in Table 1). When discussing the user experience of code reviewer recommendation, four interviewees mentioned a "notification noise" of code review requests. When code contributors create code review requests, they are intended to invite many code reviewers. As most files or directories require only one code reviewer, most CR request invitations are ignored. For example, I3 said *"Though I finish about ten code reviews each week, I may receive up to 100 code review requests. Most of these requests are ignored. Urgent requests are covered up, and contributors have to ask for reviewing code by calling or in other ways."* From the perspective of code contributors, it is reasonable as MCR asynchronously finishes code review and inviting more code reviewers can guarantee the CR process. However, it is hard for the code reviewers to prioritize code review requests on the invitation system, especially when they are overwhelmed by the notifications. Note that such a "notification noise" is not simply equivalent to a heavy code review workload. It is more likely to be caused by excessive code review invitation. I3 said *" I have to ignore the whole CR notifications and decide what to review based on my schedule and manually decide the prioritization."* Hence, we observe such noises risk making the invitation phase (S3) invalid and turning the automatic workflow back to manual. I4 mentioned that he expected the intelligent CRR can help reduce the the number of code reviewer invitaion. To improve the feasibility of CRR, practical tool should consider a trade-off between the recommended reviewer number and the CRR accuracy.

Finding 5. An excessive of invitation in the CRR system can cause "notification noise" for code reviewers, even invalidating the code review invitation process. Code reviewer recommendations should consider the issue and find a trade-off between the recommendation size and the accuracy.

We introduced the standard algorithm of CRR approaches (including the selected five approaches) during the interview. We briefly described them as "learning the historical CR-related data to predict and recommend reviewers". We referred to it as an "machine-learning-based code reviewer recommendation" and asked their opinions and expectation. All interviewees showed positive attitudes towards these approaches and expected it could help improve the efficiency when finding code reviewers. However, some interviewees pointed out that we need more efforts to make such an "machine-learning-based CRR" useful in practice. Precisely, though the CRR performance of the selected approaches is acceptable as discussed in RQ1, retrospective analysis can only reflect the history. In practice, various code review situations should be considered when recommending code reviewers. Sometimes code contributors expect the code can be merged as quickly as possible, which is also an (implicit) of many approaches in literature. However, in other situations, contributors may expect to gain knowledge from the code review process. For example, I3 said, "I do not always need the recommendation learned from historical data. When I want my code to be improved during code review, I prefer a senior professional to give high-level suggestions instead of reviewers who are already familiar with the code."

In particular, the implementation of code reviewer recommendations matters. Five interviewees gave special care about the speed

of the machine-learning-based CRR approaches. The training speed and the inference speed (recommendation speed) can impact the user experience. A good reviewer recommendation should be designed in a non-invasive way that interrupts the code review.

> **Finding 6.** Even though practitioners are confident about the machine-learning-based CRR approaches, a practical CRR system should consider various situations and works in a non-invasive way.

When discussing the CRR algorithm improvement, most interviewees (5) mainly focused on provided suggestions about including more CR-related information based on their experience and intuition. Their suggestions include (1) the time recency (e.g., *"newer modified reviewers are likely to be active"*), (2) improving new files' weights and frequency, (3) learning similar file path and code changes, and (4) considering the reviewers' social networks. Even though the suggested information are all covered in literature [2], such suggestions provide empirical evidence of adopting these data in the algorithm.

All interviewees suffer from unfamiliar reviewers as they could get limited information on from the system. Therefore, an expectation of code reviewer recommendation is bridging the information gap between code contributors and code reviewers, especially when they are not familiar with each other. For example, all interviewees agreed that showing recommending reasons for code contributors could help find appropriate reviewers. Similarly, I6 expressed that code reviewers also expected more information about the CR requests in turn.

> **Finding 7.** Practitioners expect the CRR system considers as much code review related information as possible. CRR system should help code contributors and code reviewers get more information about each other to facilitate the reviewer selecting process.

5 DISCUSSION

Code reviewer recommendation is an essential step in the whole code review workflow. Automating such a process can have an impact on the code review process in turn. Therefore, we discuss such implications of CRR approaches on the code review. Figure 3 shows the results of the statement agreements (the **Part IV** as in Table 1). We discussed all statements under the background of CRR. Interviewees expect to find reviewers who are familiar with the code changes, and they can also help gain knowledge and improve their ability. Inner-source practice does challenge the CRR as it involves more developers across different projects. Besides, reviewers also expect CRR to alleviate the review burden by helping reduce the invitation numbers (if accurate enough).

Different effectiveness of CRR approaches between proprietary and open-source projects. In RQ1, we conclude the selected approaches do not perform as well on the original reports. To avoid bias caused by experimental settings, We applied the selected approaches on originally used open-source projects Android, OpenStack, LibreOffice, and QT. We use the Mann-Whitney's U test [16] in terms of MRR, accuracy, precision, and recall on review requests of two kinds of projects. The results confirm that the difference in performance on Tencent and open-source projects are statistically

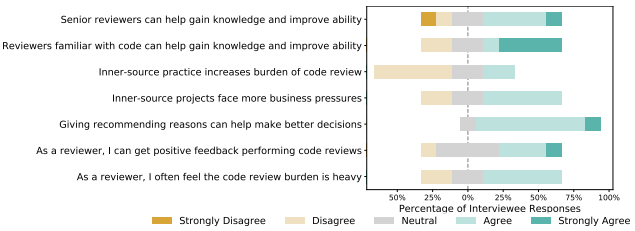

Figure 3: Statement agreements of the CRR impact.

significant, with a p-value close to zero. These replication experiments substantiate the selected existing approaches cannot perform well on proprietary projects. The inner-source brings the needs of code reviewing in touch with different unfamiliar developers. Even though the inner-source environment is similar to open-source, inner-source projects still have a business burden compared to open-source projects. Through offline experiments, we find that the metrics are not as high as those on open-source projects. It reveals the need of applying CRR approaches. Furthermore, the code reviewer recommendations can improve the scalability of the projects. Specifically, when the team is small or contributor-reviewer relationships are relatively stable, the algorithm does not seem to be necessary. However, the previous tools cannot support the complicated code reviewing environment when the number of team members grows. For example, TIE can learn the newly added reviewers dynamically to improve the scalability.

Evaluation limitation of retrospective analysis. When evaluating a CRR approach, to gain oracle conveniently, researchers follow an (often implicit) assumption that those who participated in the code review were the best developer to review the change, and those who were not invited were not appropriate reviewers [2]. We find the assumption can be valid when the purpose of code review is to pass the CR as soon as possible for subsequent tasks (e.g., merging or testing). However, when the purpose is gaining knowledge or improving code in a high-level by code review, it is hard to say the assumption can meet the needs. As for metrics, MRR cannot reflect practice well because developers are intended to add more candidates when they are not sure about the reviewers (as discussed in "notification issue"). However, when applying CRR approaches, it can be a good indicator to determine the recommendation number (instead of invite as much developer as possible).

6 THREATS TO VALIDITY

A threat to validity is that this paper only performs analysis in one company. However, Tencent is an international IT company with more than 80 thousand employees and covers many essential areas. Our samples cover different businesses, including games, instant communication, cloud, and so on. Such variety leads to different patterns of software development and code review environment. We believe such variety can guarantee the analysis can cover scenarios and the generality of our implications.

Another threat to validity is that our interviewees may not fully understand code reviewer recommendations or our questions well as developers are familiar with the role of "contributor" or "reviewer". The recommendation system is in the backend and works

as a black box for them. Their responses may introduce noise to the interview results. To reduce this threat, we briefly describe the recommendation scenarios and introduce the standard recommendation algorithm before a formal interview to ensure interviewees can appropriately understand the whole interview topic. Hence, a more comprehensive oracle of code reviewer recommendation is required to address the shortcomings of existing evaluation.

7 CONCLUSION AND FUTURE WORK

In recent years, code reviewer recommendation has been a popular research topic, helping modern code reviews better match code contributors and code reviewers. This paper performs an empirical analysis of existing CRR approaches in the international giant IT company Tencent. Based on the retrospective results, we further conducted interviews with practitioners about the expectations of CRR in practical environment. We show the findings, present implications, and discuss future direction of applying CRR approaches in practice.

In future, we plan to address the algorithm challenges discussed in the retrospective analysis. We also plan to deploy a recommendation system incorporating the implications and suggestions provided by the practitioners.

8 ACKNOWLEDGEMENT

The authors showed their sincere appreciation for all Tencent developers who interviewed with us as well as their precious opinions on the code reviewer recommendation practice.

REFERENCES

[1] Wisam Haitham Abbood Al-Zubaidi, Patanamon Thongtanunam, Hoa Khanh Dam, Chakkrit Tantithamthavorn, and Aditya Ghose. 2020. 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*. ACM, Virtual USA, 21–30.

[2] Sumit Asthana, Rahul Kumar, Ranjita Bhagwan, Christian Bird, Chetan Bansal, Chandra Maddila, Sonu Mehta, and B. Ashok. 2019. 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, Tallinn Estonia, 937–945.

[3] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, San Francisco, CA, USA, 712–721.

[4] Ricardo Baeza-Yates and Berthier A. Ribeiro-Neto. 2011. *Modern Information Retrieval - the concepts and technology behind search, Second edition.* Pearson Education Ltd., Harlow, England. http://www.mir2ed.org/

[5] JesúS Bobadilla, Fernando Ortega, Antonio Hernando, and Jesús Bernal. 2012. A collaborative filtering approach to mitigate the new user cold start problem. *Knowledge-based systems* 26 (2012), 225–238.

[6] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative research in psychology* 3, 2 (2006), 77–101. Publisher: Taylor & Francis.

[7] M. E. Fagan. 1999. Design and code inspections to reduce errors in program development. *IBM Systems Journal* 38, 2.3 (1999), 258–287. Conference Name: IBM Systems Journal.

[8] Joseph L. Fleiss. 1971. Measuring nominal scale agreement among many raters. *Psychological bulletin* 76, 5 (1971), 378.

[9] Yuanzhe Hu, Junjie Wang, Jie Hou, Shoubin Li, and Qing Wang. 2020. Is There A "Golden" Rule for Code Reviewer Recommendation? : —An Experimental Evaluation. In *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, Macau, China, 497–508.

[10] Jing Jiang, Yun Yang, Jiahuan He, Xavier Blanc, and Li Zhang. 2017. Who should comment on this pull request? analyzing attributes for more accurate commenter recommendation in pull-based development. *Information and Software Technology* 84 (2017), 48–62. Publisher: Elsevier.

[11] Huzefa Kagdi, Maen Hammad, and Jonathan I. Maletic. 2008. Who can help me with this source code change?. In *2008 IEEE International Conference on Software Maintenance*. IEEE, Beijing, China, 157–166.

[12] Vladimir Kovalenko, Nava Tintarev, Evgeny Pasynkov, Christian Bird, and Alberto Bacchelli. 2018. Does reviewer recommendation help developers? *IEEE Transactions on Software Engineering* 46, 7 (2018), 710–731. Publisher: IEEE.

[13] Xuan Nhat Lam, Thuc Vu, Trong Duc Le, and Anh Duc Duong. 2008. Addressing cold-start problem in recommendation systems. In *Proceedings of the 2nd international conference on Ubiquitous information management and communication*. 208–211.

[14] Thomas D. LaToza, Gina Venolia, and Robert DeLine. 2006. Maintaining mental models: a study of developer work habits. In *Proceedings of the 28th international conference on Software engineering*. 492–501.

[15] Laura MacLeod, Michaela Greiler, Margaret-Anne Storey, Christian Bird, and Jacek Czerwonka. 2017. Code reviewing in the trenches: Challenges and best practices. *IEEE Software* 35, 4 (2017), 34–42.

[16] Henry B. Mann and Donald R. Whitney. 1947. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics* (1947), 50–60. Publisher: JSTOR.

[17] Ehsan Mirsaeedi and Peter C. Rigby. 2020. Mitigating turnover with code review recommendation: balancing expertise, workload, and knowledge distribution. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. ACM, Seoul South Korea, 1183–1195.

[18] Mohammad Masudur Rahman, Chanchal K. Roy, and Jason A. Collins. 2016. CORRECT: Code Reviewer Recommendation in GitHub Based on Cross-Project and Technology Experience. *Proceedings of the 38th International Conference on Software Engineering Companion* (2016), 222–231.

[19] Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern code review: a case study at google. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*. ACM, Gothenburg Sweden, 181–190.

[20] CACM Staff. 2019. CodeFlow: improving the code review process at Microsoft. *Commun. ACM* 62, 2 (2019), 36–44.

[21] Anton Strand, Markus Gunnarson, Ricardo Britto, and Muhmmad Usman. 2020. 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*. ACM, Seoul South Korea, 1–10.

[22] Patanamon Thongtanunam, Raula Gaikovina Kula, Ana Erika Camargo Cruz, Norihiro Yoshida, and Hajimu Iida. 2014. Improving code review effectiveness through reviewer recommendations. In *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering*. ACM, Hyderabad India, 119–122.

[23] Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Raula Gaikovina Kula, Norihiro Yoshida, Hajimu Iida, and Ken-ichi Matsumoto. 2015. 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, Montreal, QC, Canada, 141–150.

[24] Zhiyuan Wan, Xin Xia, David Lo, and Gail C. Murphy. 2019. How does machine learning change software development practices? *IEEE Transactions on Software Engineering* (2019). Publisher: IEEE.

[25] Yaqing Wang and Quanming Yao. 2019. Few-shot learning: A survey. (2019).

[26] Jian Wei, Jianhua He, Kai Chen, Yi Zhou, and Zuoyin Tang. 2017. Collaborative filtering and deep learning based recommendation system for cold start items. *Expert Systems with Applications* 69 (2017), 29–39. Publisher: Elsevier.

[27] Elaine Welsh. 2002. Dealing with data: Using NVivo in the qualitative data analysis process. In *Forum qualitative sozialforschung/Forum: qualitative social research*, Vol. 3. Issue: 2.

[28] Xin Xia, David Lo, Xinyu Wang, and Xiaohu Yang. 2015. 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, Bremen, Germany, 261–270.

[29] Haochao Ying, Liang Chen, Tingting Liang, and Jian Wu. 2016. EARec: leveraging expertise and authority for pull-request reviewer recommendation in GitHub. In *Proceedings of the 3rd International Workshop on CrowdSourcing in Software Engineering - CSI-SE '16*. ACM Press, Austin, Texas, 29–35.

[30] Yue Yu, Huaimin Wang, Gang Yin, and Charles X. Ling. 2014. Reviewer Recommender of Pull-Requests in GitHub. In *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, Victoria, BC, Canada, 609–612.

[31] Yue Yu, Huaimin Wang, Gang Yin, and Tao Wang. 2016. Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment? *Information and Software Technology* 74 (2016), 204–218.

[32] Motahareh Bahrami Zanjani, Huzefa Kagdi, and Christian Bird. 2016. Automatically Recommending Peer Reviewers in Modern Code Review. *IEEE Transactions on Software Engineering* 42, 6 (2016), 530–543. http://ieeexplore.ieee.org/document/7328331/

[33] H. Alperen Çetin, Emre Doğan, and Eray Tüzün. 2021. A review of code reviewer recommendation studies: Challenges and future directions. *Science of Computer Programming* 208 (2021), 102652.