Is Historical Data an Appropriate Benchmark for Reviewer Recommendation Systems?

A Case Study of the Gerrit Community

Ian X. Gauthier
McGill University
Montreal, Canada
ian.gauthier@mail.mcgill.ca

Maxime Lamothe
Polytechnique Montreal
Montreal, Canada
maxime.lamothe@polymtl.ca

Gunter Mussbacher
McGill University
Montreal, Canada
gunter.mussbacher@mcgill.ca

Shane McIntosh
University of Waterloo
Waterloo, Canada
shane.mcintosh@uwaterloo.ca

***Abstract*—Reviewer recommendation systems are used to suggest community members to review change requests. Like several other recommendation systems, it is customary to evaluate recommendations using held out historical data. While history-based evaluation makes pragmatic use of available data, historical records may be: (1) overly optimistic, since past assignees may have been suboptimal choices for the task at hand; or (2) overly pessimistic, since "incorrect" recommendations may have been equal (or even better) choices.**

In this paper, we empirically evaluate the extent to which historical data is an appropriate benchmark for reviewer recommendation systems. We replicate the CHREV and WLRREC approaches and apply them to 9,679 reviews from the GERRIT open source community. We then assess the recommendations with members of the GERRIT reviewing community using quantitative methods (personalized questionnaires about their comfort level with tasks) and qualitative methods (semi-structured interviews).

We find that history-based evaluation is far more pessimistic than optimistic in the context of GERRIT review recommendations. Indeed, while 86% of those who had been assigned to a review in the past felt comfortable handling the review, 74% of those labelled as incorrect recommendations also felt that they would have been comfortable reviewing the changes. This indicates that, on the one hand, when reviewer recommendation systems recommend the past assignee, they should indeed be considered correct. Yet, on the other hand, recommendations labelled as incorrect because they do not match the past assignee may have been correct as well.

Our results suggest that current reviewer recommendation evaluations do not always model the reality of software development. Future studies may benefit from looking beyond repository data to gain a clearer understanding of the practical value of proposed recommendations.

I. INTRODUCTION

Software repository data is an invaluable resource that accumulates as a software project ages. Version Control Systems (VCSs) store *commits* that record changes to codebases. Issue Tracking Systems (ITSs) accrue *issue reports* that describe product defects, enhancement requests, and other potential improvements. Review Tracking Systems (RTSs) archive the *peer code reviews* that take place during software development.

Mining Software Repositories (MSR) approaches aim to uncover and communicate insights from historical software repository data to support knowledge acquisition and future stakeholder decisions. A popular mechanism for communicating those insights is through *recommendation systems* [34]. For instance, recommendation systems have been proposed to aid practitioners with code navigation [11], [45], task prioritization [15], [16], and task assignment [2], [37].

Reviewer recommendation is a popular and recent variant of the task assignment problem, where the recommendation system provides a list of potential reviewers (ordered by their relevance) for newly submitted change requests to an RTS.

It is challenging to evaluate (reviewer) recommendation systems in software engineering settings [13]. An optimal evaluation would be: (1) *reproducible*, allowing the research community to compare recommendation approaches in an impartial manner; (2) *meaningful*, improving the practical implications of the evaluation; and (3) *pragmatic*, making use of available data and resources to the largest extent possible. Maximizing all three of these characteristics with a single type of evaluation is impractical, since they present inherent trade-offs. For example, a highly meaningful evaluation would invite active stakeholders to weigh in on the suggestions that are generated by the recommendation system. However, such an evaluation would not be pragmatic, since recruiting a large number of participants for such a study is difficult.

Broadly speaking, recommendation systems in software engineering are often evaluated using held out historical data [22]. After being trained using a subset of the available data (the training corpus), the recommendation system is tested using another subset that was not used for training (the testing corpus). In the context of reviewer recommendation systems, the correct answer that the system is expected to generate is the list of reviewers who appear in the historical data.

The optimal reviewer recommendation system according to a history-based evaluation suggests exactly (and only) what happened in the past. Since past decisions may not have been optimal, relying on such a solution would encourage stakeholders to repeat past mistakes. Moreover, Kovalenko *et al.* [24] argue that recommending popular past reviewers provides limited value in proprietary settings, where such reviewers are often well known. In our estimation, solely relying on historical records to evaluate reviewer recommen-

dation systems suffers from two potential limitations. First, the evaluation may be *overly optimistic* since the solution that stakeholders applied may have been a suboptimal choice. Second, the evaluation may be *overly pessimistic*, since the "incorrect" suggestions may have been as appropriate (if not more so) than the solution that stakeholders had applied.

If history-based evaluations are overly optimistic or pessimistic, performance scores will overestimate or underestimate the value that the reviewer recommendation system provides. Therefore, we pose the following central question:

Is historical data an appropriate benchmark for reviewer recommendation systems?

As a concrete first step towards addressing this question, in this paper, we perform a case study of reviewer recommendation in the context of the GERRIT open source community. We conduct our study by (a) replicating the CHREV [44] and WLRREC [1] reviewer recommendation approaches, and (b) applying them to 9,679 review records spanning the last two years of GERRIT development.

We classify the examples in the history-based evaluation into "correct" and "incorrect" recommendations. Using representative samples from each category, we issue personalized questionnaires to reviewers in the GERRIT community.[1] These questionnaires asked reviewers to assess their comfort level with both reviews to which they were assigned (i.e., "correct" examples) and reviews to which they were not assigned but were recommended by a reviewer recommendation approach (i.e., "incorrect" examples). By triangulating observations from this quantitative data with qualitative data collected from semi-structured interviews with GERRIT stakeholders, we address the following research questions:

(RQ1) Optimism: How often are past assignees that were deemed correct actually suboptimal?
We find that, in the vast majority of cases, reviewers who were deemed to be "correct" were actually comfortable reviewing the given change. Indeed, 86% of reviews received comfort level ratings of either 'high' or 'expert'. Conversely, only 8% of the reviews received a comfort level rating of either 'low' or 'not comfortable'. This suggests that history-based evaluations of reviewer recommendation are *not* overly optimistic.

(RQ2) Pessimism: How often are incorrect recommendations actually appropriate?
We find that 74% of reviews that were initially deemed "incorrect" received a comfort level rating of either 'high' or 'expert' from respondents. Only 10% of reviews received a comfort level of either 'low' or 'no comfort'. These findings suggest that history-based evaluations of reviewer recommendation *are* overly pessimistic.

[1]We obtained Institutional Review Board (IRB) approval for this study from the McGill Research Ethics Board-1 (REB # 382-0218).

Broadly speaking, our results suggest that the current method of evaluating reviewer recommendation systems is far more pessimistic than optimistic. Indeed, history-based performance values are likely to be an under-approximation of the true value that reviewer recommendation system suggestions would provide to the GERRIT community. In one sense, this is a positive development, suggesting that in general, the performance of reviewer recommendation systems may be under-approximated in the literature. In another sense, under-approximated performance scores in the literature may lead practitioners to draw incorrect conclusions about the quality of current reviewer recommendation systems.

The remainder of the paper is organized as follows: Section II situates this work with respect to the literature on recommendation systems in software engineering. Section III describes the design of our case study, while Sections IV and V present the results. Section VI discusses the broader implications of our study results. Section VII discusses threats to the validity of our study. Finally, Section VIII draws conclusions and proposes directions for future work.

II. BACKGROUND AND RELATED WORK

In this section, we provide an overview of recommendation systems in software engineering generally (Section II-A) and reviewer recommendation specifically (Section II-B).

A. Recommendation Systems in Software Engineering

At their core, most recommendation systems [34] (i.e., systems that suggest to stakeholders future actions to take based on patterns or trends that have been mined from historical data) operate in a similar manner. A corpus of historical examples (i.e., the training corpus) is provided as input to a supervised or unsupervised data mining approach (e.g., statistical or machine learning regression, clustering, or classification techniques). This step produces a *model*, which can produce recommendations for future examples. While recommendation systems take a variety of forms, in our estimation, the content discovery and decision support variants, which we describe below, have received plenty of attention within the software engineering domain [16], [19], [21], [27], [40], [45].

Content Discovery Recommendation Systems. Content Discovery Recommendation Systems (CDRSs) are at the core of solutions, such as bug localization [40], where available information about a bug (e.g., issue report content) is provided as a query to an engine that searches for matches in the code base. CDRSs have also been proposed to guide developers as they make changes to a code base, suggesting other relevant locations to view/modify based on historical co-change tendencies [45] or the recency of viewing/modification [21].

Decision Support Recommendation Systems. Software practitioners make decisions to balance trade-offs regularly. Although these decisions often impact their organizations, it is not uncommon for decisions to be made based on intuition. Decision Support Recommendation Systems (DSRSs) aid practitioners in making more data-grounded decisions. Broadly speaking, these DSRSs fall into two categories.

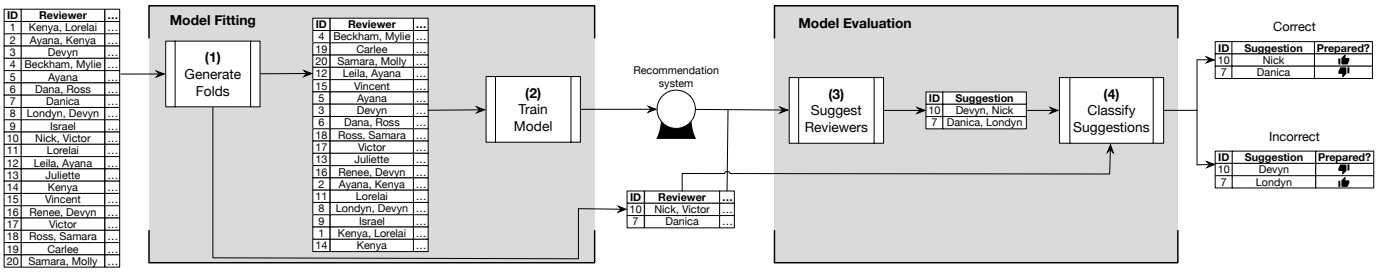

Fig. 1. An overview of how historical data is split into testing and training corpora for evaluating a recommendation system.

Effort prioritization DSRSs help practitioners to allocate constrained resources in a cost-effective manner. For example, defect prediction models [19] can be used to prioritize the testing of areas of the code base that are: (a) most likely to be defective [19]; (b) estimated to have the most defects in the future [16], or (c) estimated to have the highest defect density [27]. Moreover, effort prioritization DSRSs have been proposed to triage notifications that require stakeholder attention, such as posts on a mailing list [17], updates to issue reports [30], or review requests on code review platforms [39].

Expert assignment DSRSs strive to support stakeholders in the allocation of tasks to qualified personnel. Expert assignment DSRSs have been proposed to suggest: (a) team members to whom new issue reports should be assigned [2]; (b) experts who may be able to answer questions on developer Q&A forums [9]; and (c) reviewers to whom changelists should be assigned for code review [37].

Evaluation of Recommendation Systems. The evaluation of recommendation systems is not simple. Since future examples are not available and *in situ* evaluation is expensive and impractical, recommendation systems in software engineering are often evaluated using a corpus of held out historical data. Figure 1 shows how a corpus of changes is split into training and testing corpora for evaluating a reviewer recommendation system. A subset of the data (in the case of the figure, 10%) is held out of the training corpus at the beginning. The training corpus is then used to prepare the recommendation system. Each example in the testing corpus is then fed to the recommendation system to gather its recommendations. If the recommendations do (not) appear in the actual list of reviewers, the change is considered "correct" ("incorrect").

However, the examples labelled "correct" (i.e., the recommendations that appear in the actual list for a given change) may not have been an optimal assignee for the task. Likewise, the examples labelled "incorrect" (i.e., the recommendations that did not appear within the actual list) may have been just as capable of performing the task as the actual assignee(s). This leads to four possible categories: those considered "correct" and well suited for the task (Nick in Figure 1); those considered "correct" but not well suited for the task (Danica in Figure 1); those considered "incorrect" and not well suited for the task (Devyn in Figure 1); and those considered "incorrect" but actually well suited for the task (Londyn in Figure 1).

B. Reviewer Recommendation

In recent years, a lightweight and flexible variant of the practice of code review has become popular [7], [33]. Unlike the rigid formal code inspections of the past [12], modern practices embrace the asynchrony of global software development teams [5]. This lightweight variant, often referred to as Modern Code Review, revolves around a change-based model [4]. For each change to a system, team members are asked to critique the premise, structure, and content. Investment in this review process has been shown to pay off in qualitative [6], [7] and quantitative ways [26], [29], [33], [38]. Assigning ill-suited reviewers to review a change can lead to suboptimal review outcomes, such as slow [37] or less useful feedback [8], [23].

Considerable research effort has been invested in the problem of *reviewer recommendation*—a variant of expert assignment DSRSs, which recommend appropriate personnel for review tasks. For example, ReviewBot [3] and RevFinder [37] suggest reviewers based on how often they have contributed to the lines or modules that were modified by the patch, respectively. CoRReCT [32] suggests reviewers with relevant technological experience and experience with other related projects. Tie [41] combines file location and patch content analyses to suggest relevant reviewers. Ouni *et al.* [31] treat the reviewer recommendation problem as a multi-objective optimization problem, and solve it using evolutionary algorithms. Yang *et al.* [42] further specify what role the suggested reviewer should play (e.g., managerial, technical). Yu *et al.* [43] explore reviewer recommendation in the context of GitHub. Lipcak *et al.* [25] further analyze the level of success achieved by reviewer recommendation systems for GitHub projects.

We select reviewer recommendation as an exemplar of a popular research area where history-based evaluation is customary. We use the recent CHREV [44] and WLRREC [1] approaches as concrete reviewer recommendation solutions.

Prior work has shown that history-based evaluation of reviewer recommendation approaches may yield misleading results. Indeed, Kovalenko *et al.* [24] found that, in the setting of large software organizations, team members often know who should review their code, limiting the usefulness of the recommendation list. They argue that, in such settings, there is a need to move beyond accuracy measures when assessing the value of a reviewer recommendation system. Inspired by their work, we set out to assess the agreement between historical measures and developer perceptions.

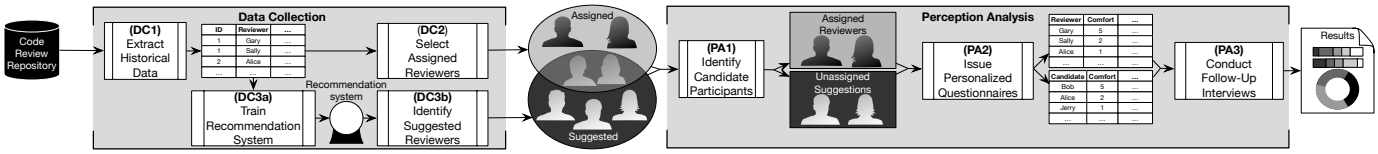

Fig. 2. An overview of our study workflow.

III. Study Design

In this section, we present our rationale for focusing our study on the Gerrit software community (Section III-A), as well as our approach to the collection (Section III-B) and analysis (Section III-C) of data.

A. Subject Community

We strive to analyze data from a large and active software community that actively invests in the practice of code review. In selecting our subject community, we identified three important criteria that needed to be satisfied:

- **Criterion 1: Track record of code review data.** To perform a robust history-based evaluation of a reviewer recommendation system, a large amount of data about prior reviews must have accrued.
- **Criterion 2: Traceable code review process.** Reviewer recommendation systems often rely upon rich historical context to produce suggestions. For example, RevFinder [37], ReviewBot [3], and xFinder [18] require access to version control history to compute code tendency and experience heuristics. Thus, the community that we select must provide review records that are well linked to the commits on which they were performed.
- **Criterion 3: Responsiveness.** Our perception analysis relies on soliciting a large quantity of high quality data from the studied community. An unresponsive community may not provide a reliable and complete perspective.

We have found that the Gerrit community—and by extension the data derived from the Gerrit project—is able to satisfy all of our evaluation criteria. In terms of Criterion 1, the Gerrit project completed 9,847 reviews from January 2018 to December 2019. In that same period, 98% of the commits in the version control system have an accompanying review record in the review tracking system, satisfying Criterion 2. Finally, since the last author has fostered professional relationships and an awareness of code review research within the Gerrit community, and because Gerrit is a code review platform, we believed that the Gerrit community would be amenable to and welcoming of our study, which would in turn lead to a strong response rate, satisfying Criterion 3.

B. Data Collection

The first step in our study is to collect data from the Gerrit community. Figure 2 shows that the process involves the extraction of historical data (DC1), from which the assigned reviewers can be selected (DC2). We can use the historical data to train (DC3a) and evaluate (DC3b) our selected reviewer recommendation systems. We describe each step below.

TABLE I
An overview of the yearly review activity of the Gerrit community. The years with the shaded cell background are selected for our analysis.

Year	# Changes	# Reviewed	Percent Reviewed	# Unique Reviewers
2012	1,395	1,128	90.86%	80
2013	3,324	2,877	86.55%	120
2014	2,607	2,201	84.43%	93
2015	2,983	2,569	86.12%	117
2016	4,426	4,069	91.93%	133
2017	5,184	4,407	86.85%	143
2018	5,074	4,992	98.38%	140
2019	4,773	4,687	98.20%	153
Total (analyzed)	9,847	9,679	98.29%	224

(DC1) Extract Historical Data. To obtain a corpus of data with which to train and test our recommendation systems, we first set out to extract historical review data from the Gerrit community. The community practices "dogfooding" by using the Gerrit software that the community develops. Thus, we first set out to extract data from the community's Gerrit instance using the REST API.[2] For each change, the extracted data includes: (a) a timestamp; (b) the author and reviewers; and (c) the impacted files and lines.

Table I provides an overview of the extracted Gerrit data. The table shows that the community has been growing over the years, attracting more than 4,000 contributions per year since 2016. In addition, during this time, a very large proportion of the changes (>85%) made to the Gerrit system can also be linked to a review reported within the system. While this high proportion of linked reviews would allow us to use all of the changes and reviews available within the Gerrit system, we choose to use only the data from 2018 and 2019.[3] Similarly to prior work [20], we consider that more recent changes indicate more familiarity with a given change. Indeed, prior work [20] has posited that the experience of a developer with regards to a given change should be weighted by its age (e.g., developer experience for a change twice as old would be halved). We therefore seek to solicit feedback from participants about data from recent changes. We use a two-year period to balance the tradeoff between the quantity of data and limiting experience degradation, since it may be difficult for the participant to recall their experience for older changes. Thus, we choose to focus our historical analysis on the reviews that occurred during this two-year period.

[2]gerrit-review.googlesource.com

[3]The bulk of the data collection and analysis for this work was conducted in 2020. Hence, data from 2020 was not included.

(DC2) Select Assigned Reviewers. History-based evaluation of reviewer recommendation systems requires the list of team members who reviewed each change. To obtain this list, we first extract the user IDs listed as reviewers of each change.

We apply both reviewer and review filtering to clean the raw extracted lists of reviewers prior to training our recommendation systems. First, we remove users from reviewer lists who are known to be accounts that represent automated bots. For example, the GerritCI account is associated with the GERRIT Continuous Integration system, not a human reviewer. The first author also manually verified the emails, first names, and last names within the data to merge any accounts that could belong to the same individual; no such instances were detected. Second, Table I shows that a small percentage (1.71%) of the changes made to the system during the study period were not reviewed. Since these changes cannot be used to train or test our recommendation system, we filter them out of our corpus. A corpus of 9,679 reviews spanning 224 unique reviewers survives the filtering process.

(DC3a) Train Recommendation System. There have been several reviewer recommendation systems proposed in the literature (see Section II). We choose to re-implement CHREV [44] and WLRREC [1], since they are recently proposed, high performance recommendation approaches.

The CHREV approach produces suggestions by computing a user score for each file in the system based on how frequently the user has participated in reviews containing the file and how recently they have done so. Users are given a score based on the proportion of the total number of comments on a file (across all reviews) that the user has made. Users are also scored based on the number of days in which they have made a comment to a file as a proportion of the total number of days in which the file has received review comments. Finally, users are scored based on the most recent day that they commented on a file during a review relative to the most recent comment made to that file by any user.

Meanwhile, WLRREC uses multi-objective optimization to consider five input measures: code ownership, reviewing experience, patch author familiarity, review participation rate, and a metric representing the review workload. The first four measures are maximized to increase the chance of participating in a review, and the last measure is minimized to reduce workload. WLRREC recommendations are not ranked. More details for each approach can be found in the original CHREV [44] and WLRREC [1] papers.

(DC3b) Identify Suggested Reviewers. For CHREV, the user scores are used to identify reviewers to suggest. For each review in the testing corpus, expertise and recency scores are computed for each potential reviewer and a ranking is produced. For WLRREC, the output is a list of potential reviewers, which are all considered to be as equally valid.

To evaluate the performance of these recommendation systems, we aim to compare the history-based evaluation scores reported in the literature with those that our models achieve. To do so, we need to split our historical data into training and testing corpora. The selection of the testing corpus is critical because it is the basis from which we will sample reviewers for our perception analysis. We allocate the latest 1,000 changes to the testing corpus (~10% of the data set). Rather than randomly sampling reviews, we allocate the latest changes because this most closely matches the evaluation scenario in reality. Moreover, to avoid other impractical evaluation scenarios, only those reviews that were completed prior to the creation of the review being evaluated are used by the recommendation system. A random sampling of training and testing corpora may create an unrealistic scenario where future data (e.g., changes from 2019) is used to suggest reviewers for past events (e.g., changes from 2018).

To identify the suggested reviewers for CHREV, we need to set a cutoff value k for the ordered list of suggestions. To compare with the initial CHREV study [44], we experiment with $k = 1, 2, 3, 5$ settings (Section IV). We use the most stringent $k = 1$ setting when selecting reviewers for our perception analysis (Section V) to avoid overburdening our subject community with unnecessary survey requests.

C. Perception Analysis

After collecting the recommended and assigned reviewers, we perform our perception analysis. Figure 2 shows that the analysis is composed of identifying candidate participants (PA1), issuing personalized questionnaires (PA2), and conducting follow-up interviews (PA3). Below, we describe each step.

(PA1) Identify Candidate Participants. We begin by comparing the recommended reviewers (see DC3b) to the lists of actual reviewers (see DC2). We label each review with one of two categories. If the suggested reviewer appears in the list of actual reviewers, we label that change—and its suggested reviewer—as "correct". Alternatively, if the suggested reviewer does not appear in the list of actual reviewers, we label that change—and the suggested reviewer—as "incorrect".

This first step generates a stratified data set from which to sample, i.e., a set of 408 "correct" changes and a set of 592 "incorrect" changes. We then apply stratified sampling to each category to select a representative set of changes for further evaluation. We draw a representative sample to achieve a confidence level of 95% with a confidence interval of ±5% from each category independently. A sample size calculation indicates that we need 198 "correct" changes and 233 "incorrect" changes.

To satisfy our sample size, we began by randomly selecting changes from each of the categories. However, because a small number of reviewers perform a large proportion of the reviews, we found that random sampling would lead to a sample that over-represented (and overburdened) the most active GERRIT reviewers. Placing too large of a burden on very active reviewers would make them less likely to participate. Moreover, it increases risk, since a lack of response from such a reviewer would produce a large gap in study data. For these reasons, we set an upper limit of 20 reviews per candidate participant. More specifically, we continue to draw samples randomly until we reach our target sample size (i.e., 198 "correct" and 233 "incorrect"), while ensuring that the

upper limit is not exceeded for any reviewer. In the end, the samples that we drew include 50 candidate participants, 26 of whom are present in both "correct" and "incorrect" lists, and three and 21 of whom are only present in the "correct" and "incorrect" lists, respectively.

(PA2) Issue Personalized Questionnaires. With the two samples of changes obtained, we create the personalized questionnaires for each candidate participant. To ensure that the study is consistent for all reviewers involved, we create a template for the questionnaire. For each change, the participants are asked how prepared they were—or would have been—to review the change on a five-point Semantic scale, whether they know of any other reviewers who might have been a good reviewer for the given change, and if they have any specific comments about the change which they felt were important to note. The questionnaire provides the option to stop and submit the questionnaire after any change in the hope that this will deter candidates from abandoning the questionnaire without submitting anything. We emailed each candidate participant individually soliciting their feedback via the personalized questionnaire. We then use the responses to these questionnaires to answer our RQs. A template for our questionnaires is available online [14].

Since reviewers with plenty of expertise will generally feel comfortable with most reviews, we solicit participation from users with varying rates of prior review participation within the GERRIT community. Figure 3(a) shows the distribution of the number of reviews that our candidate participants performed within the two-year study period. The expertise of our reviewers broadly varies, some of whom performed very few reviews (the first quartile is 34 reviews) and others performed several reviews (the third quartile is 546 reviews). We believe that this provides an accurate cross-section of the GERRIT community and reduces the likelihood of skew due to the (over)confidence of individual reviewers. We used a two-tailed Mann–Whitney U test to determine whether the distributions of both populations (i.e., reviewers who performed reviews and respondents to our survey) are equal. We obtained a p-value of 0.9442, and therefore accept the null hypothesis that the samples are not drawn from statistically distinct populations. In addition, Figure 3(b) shows the distribution of the same metric for those who responded to our questionnaire. The candidate and respondent plots are visually similar, leading us to conclude that reviewers with a similar cross-section of expertise actually participated in our questionnaire.

(PA3) Conduct Follow-Up Interviews. Having obtained the results of the questionnaire outlined within (PA2), we conduct a series of follow up interviews intended to tackle key emergent themes from within the results of the questionnaire. To this end, we reach out to a group of respondents to the questionnaire who showed interest in the content of the study and/or included responses that we felt would benefit from further elaboration. In total, we contacted nine members of the GERRIT community with six agreeing to participate—five over teleconference and one over email. Table II shows that we interviewed participants with different roles within GERRIT

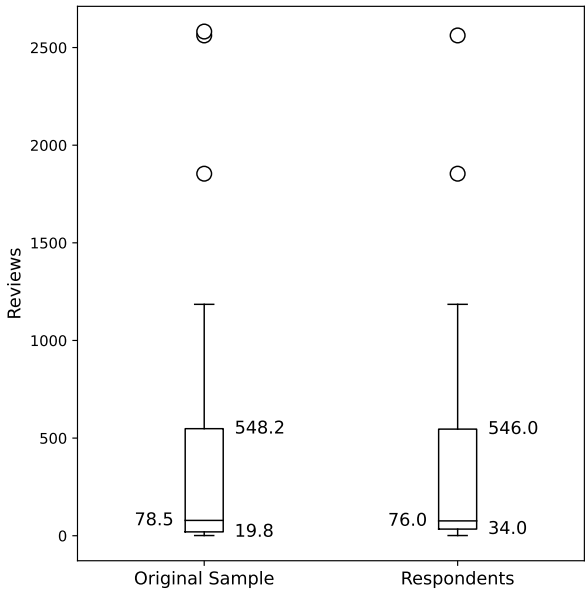

Fig. 3. The distributions of the number of reviews that had been performed in the studied period by each of the (a) candidate respondents; and (b) actual respondents. The median, first, and third quartiles are labelled.

TABLE II
AN OVERVIEW OF THE PARTICIPANTS IN OUR INTERVIEWS.

Participant	Participation Medium	Role within GERRIT
P1	Email	Maintainer
P2	Video Call	Contributor
P3	Video Call	Contributor
P4	Video Call	Maintainer
P5	Video Call	Contributor
P6	Video Call	Contributor

including maintainers and users without a specified position, experienced reviewers and new members of the community.

We apply a semi-structured approach to our interviews. The first and last authors conducted and were present for all of the interviews. An outline of key questions and interview topics were created prior to the interview session, but interviewers and interviewees had the freedom to follow (relevant) divergent topics if they arose based on responses. Questions were initially formulated based on trends in the questionnaire responses, as well as the individual interviewee responses.

The interviews were recorded, transcribed, and coded to extract themes. We use these themes to deepen the insights that we glean from our questionnaires. As such, we code the transcripts based on whether the responses relate specifically to our research questions. When responses relate directly to a research question, we check if multiple community members mention similar points regarding the same topic. In the cases in which they do, we report these findings along with the participants who stated them (participants are labeled as P1–P6 when outlining their responses) in order to corroborate or contradict the observations that we draw from the questionnaire.

IV. Preliminary Evaluation

To test if our recommendation system implementations are on par with the original implementations [1], [44], we evaluate our replicated solutions on a sample of historical data.

A. Experimental Setup

Sample size. We test both approaches on 150 Gerrit changes to mimic the testing corpus size outlined in the original cHRev paper, where three of the four systems tested had sample sizes of roughly 150 changes. Furthermore, sampling the most recent 150 changes allows for the largest possible model to be created in each case. We therefore use 150 changes to provide a fair comparison with the original paper.

Evaluation metrics. To test the recommendation systems in a comparative manner, we use the same evaluation metrics as the original cHRev paper, i.e., precision, recall, F1-score, and Mean Reciprocal Rank (MRR). For a given review x, the precision, recall and F1-score are:

$$\textit{precision}(x) = \frac{|\textit{Suggested}(x) \cap \textit{Actual}(x)|}{|\textit{Suggested}(x)|} \quad (1)$$

$$\textit{recall}(x) = \frac{|\textit{Suggested}(x) \cap \textit{Actual}(x)|}{|\textit{Actual}(x)|} \quad (2)$$

$$\textit{F1-score}(x) = 2 \times \frac{\textit{precision}(x) \times \textit{recall}(x)}{\textit{precision}(x) + \textit{recall}(x)} \quad (3)$$

where *Suggested*(x) is the list of suggested reviewers for x and *Actual*(x) is the list of team members who reviewed x.

Precision, recall, and F1-score are rank-agnostic performance measures, which ignore the position at which a reviewer is suggested. MRR is a rank-aware measure that accounts for the rank at which correct suggestions appear. The higher the rank of correct suggestions, the better (larger) the MRR value. A perfect MRR of one indicates that, for all reviews, the first suggested reviewer appears in the list of actual reviewers, while MRR values that approach zero would imply that the "correct" suggestions rank near the bottom of the list. More specifically, MRR is defined as:

$$\textit{MRR} = \frac{1}{|n|} \sum_{i=1}^{|n|} \frac{1}{\textit{rank}_i} \quad (4)$$

where n is the number of reviews on which the recommendation systems are evaluated and $\textit{rank}_i$ is the position of the first actual reviewer in the list of suggested reviewers. Since WLRRec does not rank candidates, this metric is only computed for our cHRev implementation.

B. Experimental Results

Table III shows that our implementation of cHRev is able to achieve a similar level of performance to that of the original implementation. For all measures and studied k settings, our implementation achieves performance scores within ± 5 percentage points of those reported in the original paper, suggesting that our replication is largely successful.

Similarly, Table IV shows that our implementation of WLRRec is able to achieve similar results to those of the original implementation. The 'Actual' and 'Potential' reviewers rows reflect the different choices of ground truth data as described in the original paper [1]. Although the recall values are much larger in our 'Actual Reviewers' ground truth setting, we believe that the consistent values for all other fields suggest that our replication was successful in this case as well. We believe that the differences in results are likely due to the difference in the average size of the list of actual reviewers for each change, and the differences in datasets.

TABLE III
Comparison of cHRev on the Gerrit (Value) project compared to the original's performance on the Android platform (Δ).

k	Precision		Recall		F1-Score		MRR	
	Value	Δ	Value	Δ	Value	Δ	Value	Δ
1	0.52	+0.02	0.25	-0.02	0.34	-0.01	0.62	-0.03
2	0.40	-0.01	0.37	-0.05	0.39	-0.02	0.62	-0.03
3	0.35	0.00	0.46	-0.04	0.39	-0.02	0.62	-0.03
5	0.28	-0.02	0.56	-0.05	0.37	-0.03	0.62	-0.03

TABLE IV
Comparison of WLRRec on the Gerrit project (Value) compared to the original's performance on the Android platform (Δ).

Evaluation Type	Precision		Recall		F1-Score	
	Value	Δ	Value	Δ	Value	Δ
Actual	0.15	-0.05	0.71	+0.41	0.25	-0.03
Potential	0.31	0.0	0.49	-0.01	0.38	+0.03

We find that 63% of the WLRRec recommendations overlap with the cHRev recommendations with the $k = 1$ setting. Since higher k settings reduce the overlap (59% (k=2), 56% (k=3), 51% (k=5)), we concentrate our analysis on the $k = 1$ setting. Due to the large overlap in results, and because the cHRev results are ranked and therefore allow us to target specific developers from within the recommendation list, we choose to use the recommendations from cHRev to select participants for our questionnaires and interviews.

V. Study Results

In this section, we describe the results of our study with respect to our research questions. For each question, we present our rationale, followed by our approach for addressing the question, and finally, we present the results.

(RQ1) Optimism: How often are past assignees that were deemed correct actually suboptimal?

Motivation. When performing an evaluation of the correctness of a reviewer recommendation system, researchers often count a recommendation as correct when the reviewer suggested by the system for a past change actually performed the review. However, the reviewers who reviewed past changes may not be the best choice to perform the review. For example, reviewers may participate in a review to learn about a module [7] or to cover for the optimal reviewer who is unavailable at the time of the review. These non-technical factors may introduce noise in the "correct" signal recorded in past review participation. This

TABLE V
AN OUTLINE OF THE FREQUENCY OF RESPONSES FOR BOTH "CORRECT" AND "INCORRECT" CHANGES

Comfort score	Comfort score frequency	
	"Correct" changes	"Incorrect" changes
1	1	1
2	5	5
3	5	9
4	13	22
5	52	21

type of noise would lead history-based evaluations to be overly optimistic, over-estimating the true performance of a system, since some of the cases labelled as "correct" by the evaluation may actually be incorrect. Hence, we set out to quantify and characterize the optimism of history-based evaluations.

Approach. To address RQ1, we select the sample of 198 "correct" cases, i.e., where the suggested reviewer matches a reviewer who performed a past review. We follow our sampling strategy from Section III-C (see step PA1). For each unique reviewer in our sample, we create a personalized questionnaire, which consisted of one page for each review in our sample. Each page asked the participant to (a) rate how comfortable they were when performing the review in question (on a five-point Semantic scale); (b) optionally recommend another team member who may have been able to perform the review; and (c) optionally provide their rationale for either or both of their prior responses (free-form text). We use the Semantic scale responses to gain an overview of the optimism of history-based recommendation evaluations. We then code the free-form responses using an open coding approach [35]. We use the most frequent codes to generate hypotheses about the nature of misassigned reviews. We test the validity of these hypotheses qualitatively during follow-up interviews (see step PA3 in Section III-C) and quantitatively using Spearman's ρ rank-based correlation coefficient when appropriate.

Results. We contacted 29 candidate participants of various levels of seniority within the GERRIT community. We received twelve responses (a response rate of 41%). These responses contained assessments of 76 of the 198 subject changes (38%).

It is rare for past assignees who were deemed correct to have been uncomfortable with their past reviews. Table V shows the distribution of comfort score responses. The mean comfort level for the 76 changes that received responses is 4.45. Moreover, in 65 of those 76 cases (86%), the level of confidence reported by respondents was four or five. Since our sample contains 76 instead of 198 cases, and our original population is 402 "correct" changes, with a confidence level of 95%, our confidence interval unfortunately expands to 7.03%. Nevertheless, this bodes well for history-based evaluations of review recommendation, since our responses suggest that current ground truth approaches are not overly optimistic.

The follow-up interviews further corroborate these findings. Multiple interviewees (P5, P6) mentioned that they look through a change's contents and the other reviewers already working on a change to determine whether they will bring value to a review discussion before accepting a review. One reviewer (P6) stated that they often use about a quarter of the total time spent on a change figuring out if they will provide value. This suggests that when reviewers accept a review, they are rarely a bad candidate for evaluating it.

Comfort score values are difficult to estimate. In their free-form responses, respondents often mentioned that they consider the change size and its type (e.g., merge vs. normal commit) when considering whether they would be an appropriate reviewer. Thus, we tested for correlations between size heuristics (number of changed lines, files) and change type (i.e., merge commit or not) and the comfort scores reported. For the number of lines changed and files changed, we observe weak-to-negligible correlations of $\rho = -0.12$ ($p = 0.31$) and $\rho = -0.29$ ($p = 0.01$), respectively. While the number of files has a statistically significant correlation ($p < 0.05$), its magnitude (ρ) is considered weak [36]. Finally, 18 of the 76 changes with a response were merge commits. The mean comfort score for merge commits was actually slightly higher (4.58) than non-merge commits (4.41). We perform an unpaired two-tailed Mann-Whitney U-test (a non-parametric statistical hypothesis test) to compare the comfort scores in the merge and non-merge groups. The results do not support the rejection of the null hypothesis that both groups are drawn from the same population ($p = 0.35$). Moreover, the Cliff's delta [10] (an effect size measure) is 0.05, which indicates that the practical difference between the samples is negligible.

During follow-up interviews, all participants mentioned encountering trivial changes when filling out the questionnaire. Reviewing these trivial changes does not require the same rigour. However, the interviewees did not share a common definition of what constitutes a trivial change. Definitions touched upon (a) which components a change modifies (P1); (b) how many stakeholders are potentially impacted (P1); (c) the content of the change, i.e., whether it is routine maintenance, which can be reviewed by a larger group than new features added to the system (P6); and (d) the amount of discussion that is garnered from the community (P4). Intriguingly, two interviewees felt that merge changes are generally trivial to review (P2, P4), while another interviewee felt that merge commits are rarely trivial and often required a careful review to catch potential regressions (P5). This general belief that a set of trivial changes exists without a clear definition converges with our findings from the questionnaire. Indeed, many of the reviewers who participated in the questionnaire felt very comfortable reviewing a large set of changes, but a clear pattern of what caused certain changes to be more broadly "reviewable" than others did not emerge.

> Reviewers tend to carefully select the reviews that they accept. Thus, reviewers were often highly comfortable with reviews that reviewer recommendation evaluations would deem "correct" recommendations. This suggests that optimism is not a serious problem in the context of the evaluation of reviewer recommendation systems.

(RQ2) Pessimism: How often are incorrect recommendations actually appropriate?

Motivation. Reviewer recommendation evaluations often consider a recommendation to be incorrect when the reviewer recommended by the model did not review the change. However, the results from RQ1 suggest that reviewers who are knowledgeable about a system will likely have many changes for which they are the most qualified reviewer. Since they must balance reviewing time with other tasks, reviewers may not be able to participate in the review process of every change for which they are suitable. Thus, relying solely on the reviewer who performed the review as the ground truth may underestimate the value of a reviewer recommendation model. To assess this, we set out to study if the ground truth data for reviewer recommendation systems is too pessimistic.

Approach. Similar to RQ1, we contact the reviewers whose changes had been sampled (see Section III-C). However, unlike RQ1, the changes that we sampled for RQ2 are changes for which the top-ranked reviewer suggested by the recommendation system did not perform the review. As was done for RQ1, we apply an open coding procedure to code the free-form survey results and then perform quantitative and qualitative evaluations of the most commonly occurring themes using correlation analyses and follow-up interviews, respectively.

Results. We contacted 47 reviewers (16 (34%) responded). The responses cover 58 of the targeted 233 changes (25%).

A large proportion of the recommended reviewers that are labelled as "incorrect" would have been comfortable assessing the given change. Table V shows the distribution of the comfort score responses. The mean comfort score for the 58 changes is 3.98. Moreover, 43 of 58 (74%) received a comfort score of 4 or 5. Unfortunately, since our sample size includes 58 instead of 233 changes, with an initial population size of 598 "incorrect" changes and a confidence level of 95%, our confidence interval increases to 10.74%. Nonetheless, the responses still suggest that a large proportion of "incorrect" reviewer recommendations are actually reviewers who would have been comfortable to assess a given change.

The follow-up interviews suggest that reviewers often have reasons other than their comfort level for not performing reviews. Three interviewees (P1, P2, P5) mentioned that they are quite often not able to review all of the changes they wish to due to time constraints. These interviewees emphasized that a review takes a considerable amount of time to perform well and thus, when there are a large number of changes within their area of expertise, they will not be able to review all of them. This suggests that the current approach to building a ground truth for reviewer recommendation is likely overlooking suitable candidates.

The difference in the comfort scores of "correct" and "incorrect" recommendations is not large. Indeed, the difference in mean comfort score between "correct" (RQ1) and "incorrect" (RQ2) cases is 0.47. Moreover, the difference in the proportion of reviews for which the reviewer is comfortable (providing a comfort score of 4 or 5) is only twelve percentage points. An unpaired two-tailed Mann-Whitney U-test comparing the comfort scores in the two groups indicates that there is a statistically significant difference ($p = 0.004$); however, the Cliff's delta [10] is considered small ($\delta = 0.31$). While these are non-negligible differences, their difference in magnitude is exaggerated in the evaluation of reviewer recommendations, where one case will be considered correct and the other incorrect.

The typical ground truth approaches do not account for the way in which reviews are handled in practice. Four of the six interviewees (P1, P4, P5, P6) mentioned that the choice to review a change is not solely about having complete knowledge of a change's content. They involve themselves in reviews to force themselves to learn more about the codebase or to steer the evolution of the project. Treating a reviewer as "correct" or "incorrect" may overlook such nuanced reasons for a community member's participation on a review.

Review size does not share a significant correlation with the comfort score. Similar to RQ1, we find participants often mentioned that size was a determining factor in their comfort level. However, when we measured the correlation between common size heuristics and the reported comfort score, we do not observe any strong quantitative evidence of that relationship. Indeed, the number of lines of code changed by a patch and the comfort score share a negligible, statistically insignificant negative correlation ($\rho = -0.05$, $p = 0.73$). Moreover, the number of files changed and the comfort score share a weak, statistically insignificant negative correlation ($\rho = -0.17$, $p = 0.21$).

> Recommended reviewers are often considered incorrect when the suggested reviewer did not perform the review. However, we find that in the majority of such cases, the recommended reviewers were actually quite comfortable to review those changes. Indeed, the difference in the comfort scores of "correct" and "incorrect" recommendations is small.

VI. Practical Implications

In our estimation, our study has three key implications for researchers in the areas of reviewer recommendation and recommendation systems for software engineering.

A. Reviewer Recommendation

Current evaluation procedures systematically underestimate the performance of reviewer recommendation systems. Our findings indicate that reviewers who performed the review rarely felt uncomfortable with the task (RQ1). This suggests that the common ground truth for reviewer recommendation evaluations (i.e., the reviewers who performed the task) are generally sound. Perhaps more interestingly, our findings also indicate that suggested reviewers who did not perform reviews were rarely uncomfortable with those tasks (RQ2). This suggests that history-based evaluations will underestimate the performance of a reviewer recommendation system, since such cases will be labelled as "incorrect".

Recommending the "correct" reviewer may not be the best way to evaluate reviewer recommendation systems. Our participants pointed out that they participate in reviews for various reasons that do not relate to being the area expert. For example, they use assigned reviews as a forcing function to learn about an area of the codebase (RQ2). While there is a growing literature illustrating the non-technical benefits and challenges of code reviewing practices [4], [6], [7], [33], its impact on reviewer recommendation is not yet fully understood. Indeed, Kovalenko *et al.* [24] observe that recommended reviewers rarely provide value for the authors of changes. Since we observe that reviewer recommendation systems rarely suggest reviewers who are uncomfortable with the reviewing task (RQ1, RQ2), we believe that identifying the "correct" reviewer is a relatively easy target to achieve—in fact, our data suggests performance in the literature is likely being underestimated. Instead, we recommend that the next goal for reviewer recommendation systems should be to balance the reviewing workload while optimizing for team-based constraints. A recent example of such an approach was explored by Mirsaeedi and Rigby [28], who propose methods to mitigate turnover risk, i.e., the cost associated with the departure of a team member, by simulating the reassignment of reviewers to different reviewing tasks.

B. Recommendation Systems for Software Engineering

The "status quo" history-based evaluations may not be appropriate. History-based evaluations are almost ubiquitous in evaluating recommendation systems in software engineering. While our findings are specific to the reviewer recommendation problem, we suspect that they could generalize to (at least) most other expert assignment DSRSs. Nonetheless, our findings serve as an existence proof that history-based evaluations are imperfect. Based on our results, we recommend that researchers evaluate the appropriateness of a history-based evaluation using qualitative as well as quantitative evaluations [13], [22] before adopting it.

VII. Threats to Validity

Below, we discuss the threats to the validity of our study.

A. Construct Validity

Threats to construct validity jeopardize the certainty that an operationalization measures what it set out to measure. To conduct our study, we reimplemented two reviewer recommendation approaches. While we believe that they are valid reimplementations, we did not have access to the original source code and could only implement the systems based on what was described in their respective papers [1], [44]. As a result, it is possible that our reviewer recommendation systems are not exact replicas. On the other hand, our preliminary evaluations achieved performance scores that were similar to those reported in the original papers.

In addition, our sampling procedure for our questionnaire analysis has limitations. First, we chose to limit the number of reviews to which a given contributor was asked to respond. While we took measures to ensure the set of sampled changes meaningfully represents the overall set of reviewed changes, limiting the number of reviews was a hindrance. Second, we only evaluated recommendations from the $k = 1$ setting (i.e., quantity of suggested reviewers). While we understand that another k setting may be more reflective of what is done in the Gerrit community, we selected $k = 1$ to avoid overburdening our participants. We verified the influence on the results of the two approaches by testing various k values for WLRREC and CHREV and found that larger k values still contained high overlap between the two recommendation systems. Therefore, the $k = 1$ setting allowed us to reduce the burden on the community while having a limited impact on the study scope. Furthermore, each unit increase in k would require 431 more requests for information from reviewers to retain our sample size. These additional requests risked antagonizing the community, potentially reducing our overall response rate. We attempted to mitigate this issue by not only conducting a survey but also conducting developer interviews to get clear insights from the developers. We believe that this decision was warranted as we have been able to achieve a high response rate (41% of reviewers in the optimistic case and 34% in the pessimistic case participated).

In addition, several participants could only respond to a subset of the changes that we selected for them, i.e., 35% of optimistic sampled reviews and 25% of pessimistic sampled reviews received responses. While this did expand our confidence intervals (i.e., $\pm 5\%$ was expanded to $\pm$ 7.03% and 10.74%, respectively), we were still able to compute meaningful estimates. We acknowledge that our sample sizes could have been calibrated to reflect prior software engineering survey response rates. However, we believe that this threat was mitigated by our comparatively high response rate, and because we were still able to compute meaningful estimates from the data that we did obtain.

Finally, we rely on the self-assessments of the comfort levels of our respondents to estimate their suitability for tasks. Since self-assessments may not accurately reflect true ability levels, these values may be skewed.

B. Internal Validity

Threats to internal validity pertain to alternative hypotheses that could also explain the results that we observe. One such threat is that the most experienced reviewers will generally feel comfortable reviewing most changes to the GERRIT code base. We acknowledge this threat and strive to mitigate it by soliciting participation from a broad cross-section of participants with varying expertise levels from within the GERRIT community (see Section III-C).

C. External Validity

Threats to external validity have to do with the generalizability of our conclusions to other contexts. Due to the practical cost of our analysis procedure, we chose to focus our study on one community. We acknowledge that the specific characteristics of the Gerrit community may have influenced

the overall results of the study and thereby affect the generalizability of our conclusions. However, we believe that the findings presented in this paper apply to a diverse selection of developers because our questionnaire participants and interviewees are employed by several organizations including multinationals like Google, Apple, and Ericsson, as well as small to medium-sized enterprises like GerritForge. While they all participate in the GERRIT community, their perspectives are informed by the internal reviewing processes at their employer sites. Nevertheless, replication of our study in the context of other organizations may prove useful.

VIII. Conclusion and Future Work

Like any system, reviewer recommendation systems require evaluation approaches to understand how well they are able to perform. The most prominent evaluation scheme for recommendation systems in software engineering relies heavily on historical records to act as a ground truth on which to test. This ground truth may lead to incorrectly labelled recommendations, as in the context of reviewer recommendation, reviewers who reviewed past changes may not have been comfortable doing so. In addition, candidates who were well prepared to review a change may not have had the opportunity to do so.

Hence, in this paper, we investigate the following question:

Is historical data an appropriate benchmark for reviewer recommendation systems?

Through a case study of the GERRIT community, we find that the answer is no because:

- The current ground truth is overly pessimistic and often mislabels reviewers as incorrect when they would have been comfortable reviewing the change.
- The difference in comfort level of reviewers labelled "correct" and "incorrect" by the original ground truth is small (Mean comfort scores of 4.45 vs 3.98, Mann-Whitney U-test $p = 0.004$, small effect size ($\delta = 0.31$)).

These results suggest that recommending "correct" reviewers may not be a difficult hurdle to clear. This might explain why reviewer recommendations usually do not provide valuable support for developers in commercial settings [24]. However, we do not believe that reviewer recommendation systems are without an application. By pivoting the goal of such recommendation systems away from identifying "correct" reviewers to aiding with other non-technical goals, such as minimizing the workload of the core team or mitigating the risk of knowledge loss [28], we believe that reviewer recommendation systems may find a more useful niche.

References

[1] W. H. A. Al-Zubaidi, P. Thongtanunam, H. K. Dam, C. Tantithamthavorn, and 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*, ser. PROMISE 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 21–30.

[2] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of the International Conference on Software Engineering*, 2006, pp. 361–370.

[3] V. Balachandran, "Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation," in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 931–940.

[4] T. Baum, O. Liskin, K. Niklas, and K. Schneider, "A faceted classification scheme for change-based industrial code review processes," in *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, 2016, pp. 74–85.

[5] T. Baum and K. Schneider, "On the need for a new generation of code review tools," in *Product-Focused Software Process Improvement*, 11 2016, pp. 301–308.

[6] O. Baysal, O. Kononenko, R. Holmes, and M. W. Godfrey, "Investigating technical and non-technical factors influencing modern code review," *Empirical Software Engineering*, vol. 21, no. 3, pp. 932–959, 2016.

[7] C. Bird and A. Bacchelli, "Expectations, outcomes, and challenges of modern code review," in *Proceedings of the International Conference on Software Engineering*. IEEE, May 2013. [Online]. Available: https://www.microsoft.com/en-us/research/publication/expectations-outcomes-and-challenges-of-modern-code-review/

[8] A. Bosu, M. Greiler, and C. Bird, "Characteristics of useful code reviews: An empirical study at microsoft," in *Proceedings of the International Conference on Mining Software Repositories*, 2015, pp. 146–156.

[9] M. Choetkiertikul, D. Avery, H. K. Dam, T. Tran, and A. Ghose, "Who will answer my question on stack overflow?" in *Proceedings of the Australasian Software Engineering Conference*, 2015, pp. 155–164.

[10] N. Cliff, "Dominance statistics: Ordinal analyses to answer ordinal questions," *Psychological bulletin*, vol. 114, no. 3, p. 494, 1993.

[11] D. Cubranic, G. C. Murphy, J. Singer, and K. S. Booth, "Hipikat: A project memory for software development," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 446–465, 2005.

[12] M. E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Systems Journal*, vol. 15, no. 3, pp. 182–211, 1976.

[13] J. Garcia-Gathright, C. Hosey, B. S. Thomas, B. Carterette, and F. Diaz, "Mixed methods for evaluating user satisfaction," in *Proceedings of the 12th ACM Conference on Recommender Systems*, ser. RecSys '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 541–542.

[14] I. Gauthier, M. Lamothe, G. Mussbacher, and S. McIntosh, "Is historical data an appropriate benchmark for reviewer recommendation systems? a case study of the gerrit community," Aug 2021. [Online]. Available: https://figshare.com/articles/conference_contribution/Is_Historical_Data_an_Appropriate_Benchmark_for_Reviewer_Recommendation_Systems_A_Case_Study_of_the_Gerrit_Community/14473575/1

[15] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "Characterizing and predicting which bugs get fixed: An empirical study of microsoft windows," in *Proceedings of the International Conference on Software Engineering*, 2010, pp. 495–504.

[16] A. E. Hassan and R. C. Holt, "The top ten list: Dynamic fault prediction," in *Proceedings of the International Conference on Software Maintenance*, 2005, pp. 263–272.

[17] W. M. Ibrahim, N. Bettenburg, E. Shihab, B. Adams, and A. E. Hassan, "Should i contribute to this discussion?" in *Proceedings of the International Conference on Mining Software Repositories*, 2010, pp. 181–190.

[18] H. Kagdi, M. Hammad, and J. I. Maletic, "Who can help me with this source code change?" in *2008 IEEE International Conference on Software Maintenance*, 2008, pp. 157–166.

[19] Y. Kamei and E. Shihab, "Defect prediction: Accomplishments and future challenges," in *Proceedings of the Future of Software Engineering track of the International Conference on Software Analysis, Evolution, and Reengineering*, 2016, pp. 33–45.

[20] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A large-scale empirical study of just-in-time quality assurance," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 757–773, 2013.

[21] M. Kersten and G. C. Murphy, "Mylar: A degree-of-interest model for ides," in *Proceedings of the International Conference on Aspect-Oriented Software Development*, 2005, pp. 159–168.

[22] B. P. Knijnenburg, M. C. Willemsen, Z. Gantner, H. Soncu, and C. Newell, "Explaining the user experience of recommender systems," *User Modeling and User-Adapted Interaction*, vol. 22, no. 4, pp. 441–504, Oct 2012.

[23] O. Kononenko, O. Baysal, and M. W. Godfrey, "Code review quality: How developers see it," in *Proceedings of the International Conference on Software Engineering*, 2016, pp. 1028–1038.

[24] V. Kovalenko, N. Tintarev, E. Pasynkov, C. Bird, and A. Bacchelli, "Does reviewer recommendation help developers?" *IEEE Transactions on Software Engineering*, pp. 1–1, 2018.

[25] J. Lipcak and B. Rossi, "A large-scale study on source code reviewer recommendation," in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2018, pp. 378–387.

[26] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects," *11th Working Conference on Mining Software Repositories, MSR 2014 - Proceedings*, 05 2014.

[27] T. Mende and R. Koschke, "Effort-aware defect prediction models," in *Proceedings of the European Conference on Software Maintenance and Reengineering*, 2010, pp. 107–116.

[28] E. Mirsaeedi and P. Rigby, "Mitigating turnover with code review recommendation: Balancing expertise, workload, and knowledge distribution," in *Proceedings of the International Conference on Software Engineering*, 2020, p. To appear.

[29] R. Morales, S. McIntosh, and F. Khomh, "Do code review practices impact design quality? a case study of the qt, vtk, and itk projects," in *Proceedings of the International Conference on Software Analysis, Evolution, and Reengineering*, 2015, pp. 171–180.

[30] D. Mukherjee and M. Garg, "Which work-item updates need your response?" in *Proceedings of the International Conference on Mining Software Repositories*, 2013, pp. 12–21.

[31] A. Ouni, R. G. Kula, and K. Inoue, "Search-based peer reviewers recommendation in modern code review," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2016, pp. 367–377.

[32] M. M. Rahman, C. K. Roy, and J. A. Collins, "Correct: Code reviewer recommendation in github based on cross-project and technology experience," in *Proceedings of the 38th International Conference on Software Engineering Companion*, ser. ICSE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 222–231. [Online]. Available: https://doi.org/10.1145/2889160.2889244

[33] P. C. Rigby and C. Bird, "Convergent software peer review practices," in *Proceedings of the the joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC/FSE)*. ACM, August 2013, preprint available upon request to cbird@microsoft.com or peter.rigby@concordia.ca. [Online]. Available: https://www.microsoft.com/en-us/research/publication/convergent-software-peer-review-practices/

[34] M. P. Robillard, W. Maalej, R. J. Walker, and T. Zimmermann, Eds., *Recommendation Systems in Software Engineering*. Springer, 2014.

[35] A. Strauss and J. M. Corbin, *Grounded theory in practice*. Sage, 1997.

[36] R. Taylor, "Interpretation of the correlation coefficient: A basic review," *Journal of Diagnostic Medical Sonography*, vol. 6, no. 1, pp. 35–39, 1990.

[37] P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida, and K. 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)*, 2015, pp. 141–150.

[38] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida, "Investigating code review practices in defective files: An empirical study of the qt system," in *Proceedings of the International Conference on Mining Software Repositories*, 2015, pp. 168–179.

[39] R. Wen, D. Gilbert, M. G. Roche, and S. McIntosh, "BLIMP Tracer: Integrating Build Impact Analysis with Code Review," in *Proceedings of the International Conference on Software Maintenance and Evolution*, 2018, pp. 685–694.

[40] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Transactions on Software Engineering*, vol. 42, no. 8, pp. 707–740, 2016.

[41] X. Xia, D. Lo, X. Wang, and X. Yang, "Who should review this change? putting text and file location analyses together for more accurate recommendations," in *Proceedings of the International Conference on Software Maintenance and Evolution*, 2015, pp. 261–270.

[42] C. Yang, X.-h. Zhang, L.-b. Zeng, Q. Fan, T. Wang, Y. Yu, G. Yin, and H.-m. Wang, "Revrec: A two-layer reviewer recommendation algorithm in pull-based development model," *Journal of Central South University*, vol. 25, pp. 1129–1143, 05 2018.

[43] Y. Yu, H. Wang, G. Yin, and T. Wang, "Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment?" *Information and Software Technology*, vol. 74, 01 2016.

[44] M. B. Zanjani, H. Kagdi, and C. Bird, "Automatically recommending peer reviewers in modern code review," *IEEE Transactions on Software Engineering*, vol. 42, no. 6, pp. 530–543, 2016.

[45] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl, "Mining version histories to guide software changes," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 429–445, 2005.