Workload-Aware Reviewer Recommendation using a Multi-objective Search-Based Approach

Wisam Haitham Abbood Al-Zubaidi, Patanamon Thongtanunam, Hoa Khanh Dam
Chakkrit Tantithamthavorn, Aditya Ghose
University of Wollongong. The University of Melbourne. Monash University.

ABSTRACT

Background: Reviewer recommendation approaches have been proposed to provide automated support in finding suitable reviewers to review a given patch. However, they mainly focused on reviewer experience, and did not take into account the review workload, which is another important factor for a reviewer to decide if they will accept a review invitation. *Aim:* We set out to empirically investigate the feasibility of automatically recommending reviewers while considering the review workload amongst other factors. *Method:* We develop a novel approach that leverages a multi-objective metaheuristic algorithm to search for reviewers guided by two objectives, i.e., (1) maximizing the chance of participating in a review, and (2) minimizing the skewness of the review workload distribution among reviewers. *Results:* Through an empirical study of 230,090 patches with 7,431 reviewers spread across four open source projects, we find that our approach can recommend reviewers who are potentially suitable for a newly-submitted patch with 19% - 260% higher F-measure than the five benchmarks. *Conclusion:* Our empirical results demonstrate that the review workload and other important information should be taken into consideration in finding reviewers who are potentially suitable for a newly-submitted patch. In addition, the results show the effectiveness of realizing this approach using a multi-objective search-based approach.

CCS CONCEPTS

• **Software and its engineering → Search-based software engineering**.

ACM Reference Format:
Wisam Haitham Abbood Al-Zubaidi, Patanamon Thongtanunam, Hoa Khanh Dam and Chakkrit Tantithamthavorn, Aditya Ghose. 2020. Workload-Aware Reviewer Recommendation using a Multi-objective Search-Based Approach. In *Proceedings of the 16th ACM SIGSOFT International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE '20), November 8–9, 2020, Virtual, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3416508.3417115

1 INTRODUCTION

Code review is one of the important quality assurance practices in a software development process. The main goal of code review is to improve the overall quality of a patch (i.e., a set of software changes) through a manual examination done by developers other than the patch author. Recently, many software organizations have adopted a lightweight variant of code review called Modern Code Review (MCR) [1, 15, 22, 25], which focuses on collaboration among team members to achieve high quality of a software product. Several studies have shown that active and rigorous code review can decrease the number of post-release defects [17, 28]. In addition, the collaborative practice of MCR provides additional benefits to team members such as knowledge transfer [1] and sharing code ownership [29].

Effective code reviews require active participation of reviewers with related knowledge or experience [2, 23]. Several studies have shown that a patch tends to be less defective when it was reviewed and discussed extensively by many reviewers [3, 14, 28]. Furthermore, recent work has shown that active review participation can only be achieved by inviting active and experienced reviewers [31]. However, finding suitable reviewers is not a trivial task, and this has to be done quickly to avoid negative impact on the code review timeliness [32]. Hence, several studies have proposed automated approaches to support the recommendation of reviewers for a newly-submitted patch [2, 19, 32, 34, 36, 38].

Intuitively, reviewing experience should be the key factor when selecting reviewers for a newly-submitted patch [2, 32, 34, 38]. However, recent studies pointed out that requesting only experts or active reviewers for a review could potentially burden them with many reviewing tasks [7, 16]. Prior work has also shown that the length of review queues, i.e., the number of pending review requests, can lead to review delays [4]. In fact, recent work [15, 25] reported that development teams in practice take the review workload into account when selecting reviewers for a newly-submitted patch. Furthermore, Ruangwan et al. [24] showed that invited reviewers often considered their workload (i.e., the number of remaining reviews) when deciding whether they should accept new invitations. These empirical findings highlight that the reviewer workload should be considered when selecting reviewers for a newly-submitted patch.

Hence, this work conducts an empirical study to investigate the feasibility of considering the review workload amongst other factors in automatically recommending reviewers. To do so, we develop a Workload-aware Reviewer Recommendation approach called **WLRRec**. Unlike the previous reviewer recommendation approaches which mainly focus on the reviewing experience [2, 19, 32, 36, 38], our WLRRec considers a wider range of information, including the review workload. More specifically, our WLRRec recommends reviewers based on 4+1 key reviewer metrics, i.e., four metrics including code ownership, reviewing experience, familiarity with the patch author, and review participation rate, and one metric representing the review workload. We use these metrics to define two

PROMISE '20, November 8–9, 2020, Virtual, USA

ACM ISBN 978-1-4503-8127-7/20/11…$15.00
https://doi.org/10.1145/3416508.3417115

objectives: (1) maximizing the chance of participating a review, and (2) minimizing the skewness of the review workload distribution among reviewers. To find reviewers for a newly-submitted patch, our WLRRec leverages a multi-objective meta-heuristic algorithm, namely non-dominated sorting genetic algorithm (NSGA-II) [9], to search for solutions that meet these two objectives.

Through an evaluation of 230,090 patches with 7,431 reviewers spread across four large open source software projects (Android, LibreOffice, Qt, and OpenStack), our results suggest that:

(1) when considering 4+1 key reviewer metrics, our WLRRec can recommend reviewers who are potentially suitable for a newly-submitted patch with 19% - 260% higher F-measure than the five benchmarks;
(2) including an objective to minimize the skewness of the review workload distribution would be beneficial to find other potential reviewers that might be overlooked by the other approaches which focus only on reviewing experience; and
(3) the multi-objective meta-heuristic algorithm, NSGA-II, is effective in searching for reviewers who are potentially suitable for a newly-submitted patch.

Our empirical results demonstrate the potential of using a wider range of information and leveraging the multi-objective meta-heuristic algorithm to find reviewers who are potentially suitable for a newly-submitted patch.

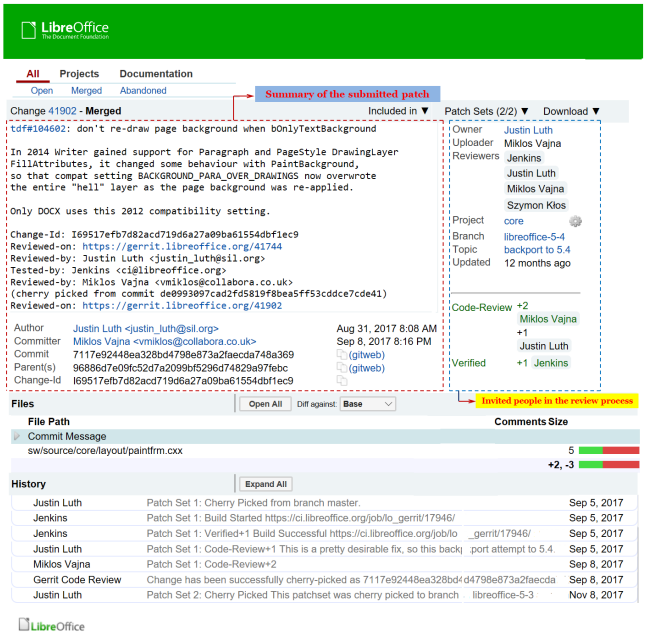

Figure 1: An example of a review in the LibreOffice Project.

2 BACKGROUND AND RELATED WORK

2.1 Modern Code Review

Modern code review is a code review process that is a light-weight variant of software inspection. The process is supported by an online code review tool (e.g., Gerrit). Recently, a modern code review process is widely used in both open source and industrial software projects [22]. Modern code review is a collaborative code review process, where developers other than the author examine a patch (i.e., a set of code changes) submitted by the author. The modern code review process typically consists of five main steps:

(1) An author uploads a patch to a code review tool.
(2) An author makes a review request to reviewers.
(3) The reviewers who accept the review request examine the patch, provide comments and a vote, where a positive vote indicates that the patch is of the quality; and a negative vote indicates that the patch needs a revision.
(4) The author revises the patch to address the reviewer feedback and uploads a new revision.
(5) If the revised patch addresses the reviewer concerns, the patch is marked as *merged* for integration into the main code repository. If the patch requires a large rework, the patch is marked as *abandoned*.

Figure 1 provides an example of a code review in the LibreOffice project. In this example, `Justin` is an author of the patch. Two additional developers (i.e., `Miklos` and `Szymon`) were invited to review the patch, while `Jenkins` is the automated Continuous Integration (CI) system. Although two reviewers were invited, only `Miklos` participated in this review by providing a vote of +2. Furthermore, the review of this patch took three days from its creation date (on September 5, 2017) to get a vote from the reviewer (on September 8, 2017). Hence, finding suitable reviewers would be beneficial to the code review process in terms of reducing the delay. However, finding reviewers who will participate in a code review is challenging as many factors can play a role, which will be discussed in the next subsection.

2.2 Related work

2.2.1 Reviewer Recommendation Approaches. Finding reviewers can be a challenging task for a patch author, especially in globally-distributed software development teams like open source software projects. Thongtanunam et al. [32] showed that the patch author of 4% to 30% of the reviews in open source projects could not find reviewers. This problem delays the reviewing process and consequently affect the software development overall. To address this problem, a number of reviewer recommendation approaches have been proposed to help a patch author find reviewers for a newly-submitted patch [2, 19, 32, 36, 38]. The key idea of those existing approaches is to find reviewers with high reviewing experience. They assume that reviewers for a newly-submitted patch should be those who have reviewed many related patches in the past. For example, Balachandran [2] proposed ReviewBot which leverages a line change history of the changed files in a newly-submitted patch to find reviewers. Thongtanunam et al. [32] proposed RevFinder which identifies the related past patches based on the file path similarity of the changed files in a newly-submitted patch. Zanjani et al. [38] has showed that the reviewing experience can change over time. Thus, they proposed cHRev which considers the frequency and recency of reviewing activities when computing reviewing experience.

Recent reviewer recommendation approaches also take into account the historical interaction between reviewers and the patch

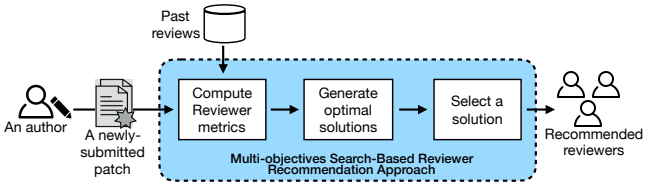

Figure 2: An overview of our approach

author. For example, Yu et al. [36, 37] built a review collaboration network to measure the co-reviewing frequency among developers. Hence, the reviewers for a newly-submitted patch are those who have high reviewing experience and high co-reviewing frequency with the author of that patch. Instead of using a heuristic approach like Yu et al. [36, 37], Ouni et al. [19] developed RevRec which uses the genetic algorithm (GA) to search for reviewers based on the reviewing experience and the review collaboration network. Their work however follows a single-objective optimization approach.

2.2.2 Reviewer Participation & Selection. While the reviewer recommendation approaches in the literature mainly focus on the reviewing experience and the historical interaction between the patch author and reviewers, several empirical studies have shown that other factors also play a role in reviewer participation. Kovalenko et al. [15] showed that when selecting a reviewer, the patch author considers a wide range of information such as knowledge, ownership, qualities of reviewers. Rigby and Storey [23] reported that in the mailing list-based code reviews, reviewers select a patch for a review based on their interests and the priorities of a patch. A survey study of Bosu et al. [7] also reported that reviewers tend to decline a review request if the patch is not in the area of their ownership or expertise. Recent empirical studies have shown that the activeness of a reviewer is a strong indicator of the reviewer participation [24, 31].

Several recent studies pointed out that reviewing workload should be considered when selecting reviewers. A survey study of Kovalenko et al. [15] reported that the reviewer workload is one of the factors that a patch author considers when selecting a reviewer. Sadowski et al. [25] reported that development teams at Google use a system that assigns reviews in a round-robin manner to take their review workload into account. Ruangwan et al. [24] also demonstrated empirical evidence that the number of remaining reviews has a negative impact on the likelihood of accepting a review request of a reviewer. Moreover, Kononenko et al. [14] showed that the length of review queue has a statistically significant impact on the review quality (i.e., whether developers catch or miss bugs during a code review).

2.2.3 Novelty. Motivated by the empirical findings of prior studies, we develop a novel workload-aware reviewer recommendation (WLRRec) approach. Our WLRRec uses 4+1 key reviewer metrics, i.e., four metrics indicating whether a review request will be accepted, and one metric representing the review workload. Unlike the previous reviewer recommendation approaches, our WLRRec is the first to take code ownership, a review participation rate, and the workload of reviewers into account. Since there are trade-offs in considering these 4+1 reviewer metrics, we employ a multi-objective search-based approach called the non-dominated sorting genetic algorithm (NSGA-II) [9]. The closest related work is RevRec of Ouli et al. [19] but they used a single-objective search-based approach (rather than a multi-objective approach as we propose here). In addition, RevRec does not consider the reviewer workload into consideration when recommending reviewers.

3 A WORKLOAD-AWARE REVIEWER RECOMMENDATION APPROACH

3.1 An Approach Overview

Figure 2 provides an overview of our Workload-aware Reviewer Recommendation approach (WLRRec) which uses a search-based software engineering (SBSE) approach. Given a newly-submitted patch, our WLRRec will first ***compute reviewer metrics*** for reviewer candidates (i.e., reviewers who have reviewed at least one patch in the past). We use five metrics which measure the experience, historical participation, and reviewing workload of reviewer candidates. Then, we employ an evolutionary search technique to ***generate optimal solutions*** where each solution is a set of reviewer candidates. The search of the solution candidates is guided by two objectives: (1) to maximize the chance of participating a review, and (2) to minimize the skewness of the review workload distribution among reviewers. The first objective considers the experience and historical participation of reviewers, while the second objective considers the review workload of reviewers. These two objectives are in conflict with each other. If we recommend only experts or active reviewers for newly-submitted patches, the reviewing workload will be highly skewed to those experts or active reviewers. On the other hand, if we only focus on establishing a highly balanced reviewing workload, the reviewers that we recommend may not be experienced and familiar with a code patch (which may affect the review quality) or active (which may delay both the review and development process).

The intuition behind the first objective is that as found in prior studies, the areas of expertise and the areas of code with which the reviewers are familiar are ones of the major reasons to accept a review request [4, 7, 16]. In addition, recent work has shown that historical participation (i.e., the review participation rate and historical interaction with a patch author) also plays an important role in the decision of accepting a review request [7, 24]. For the second objective, several studies have pointed out that making review requests to only experts or active reviewers could potentially burden them with many reviewing tasks [7, 16]. A recent work [24] also shows that the number of remaining reviews (RR) is negatively associated with the likelihood of accepting a review request. Hence, we want to ensure that our reviewer recommendation approach does not add more reviews to a particular group of reviewers (e.g., experts). MacLeod et al. [16] also suggested that requesting less experienced (but available) reviewers could potentially speed up the code review process and balance the team's workload. These motivate us to develop an approach that considers both of those two objectives at the same time.

In this work, we leverage the non-dominated sorting genetic algorithm (NSGA-II) [9] to find the optimal solutions with respect to the two objectives. The algorithm is based on the principle that

a population of solution candidates is evolved towards better solutions. At the end of the search process, the algorithm returns a set of optimal solutions, i.e., sets of reviewers that satisfy our objectives. Finally, we use a Pareto front to ***identify the optimal solution***, i.e., a set of reviewers that will be recommended for a newly-submitted patch. Below, we describe our reviewer metrics, the approaches of generating and identifying the optimal solutions in details.

3.2 Compute Reviewer Metrics

To recommend reviewers, we measure the experience, historical participation, and reviewing workload of reviewer candidates using five metrics. These metrics will be used in our fitness functions (i.e., objectives) in the multi-objective evolutionary approach. Below, we describe the intuition based on the literature and the calculation for each of our metrics.

Code Ownership (CO). CO measures the proportion of past reviews that had been authored by a reviewer candidate. Bird *et al.* [6] showed that the developer who authored many code changes should be accounted as an owner of those related areas of code. Hence, a reviewer for a newly-submitted patch should be an owner of the code that is impacted by the patch. Several studies also show that reviewers are likely to participate in the patch that they have related experience [7, 24]. Hence, we measure the CO based on the approach of Bird *et al.* [6]. More specifically, given a newly-submitted patch p for a review, we measure CO of a reviewer candidate r using the following calculation:

$$CO(r,p) = \frac{1}{|M(p)|} \sum_{m \in M(p)} \frac{\text{author}(r,m)}{c(m)} \quad (1)$$

where $M(p)$ is a set of modules (i.e., directories) in the patch p, $\text{author}(r,m)$ is the number of past reviews that were authored by the reviewer r, and $c(m)$ is the total number of past reviews that have a module m.

Reviewing Experience (RE). RE measures the proportion of past reviews that had been reviewed by a reviewer candidate. Similar to CO, a recent study showed that developers can gain expertise on the related areas of code by actively participating code reviews [30]. Several reviewer recommendation approaches are also based on the similar intuition, i.e., the appropriate reviewers are those who reviewed many similar reviews in the past [19, 21, 32, 38]. Hence, given a newly-submitted patch p for a review, we measure RE of a reviewer candidate r using the calculation of Thongtanunam *et al.* [30] which is described as follows:

$$RE(r,p) = \frac{1}{|M(p)|} \sum_{m \in M(p)} \frac{\text{review}(r,m)}{c(m)} \quad (2)$$

where $\text{review}(r,m)$ is a proportion of review contributions that the reviewer candidate r made to the past reviews K using a calculation of $\sum_{k \in K(r,m)} \frac{1}{R(k)}$ [30].

Familiarity with the Patch Author (FPA). Recent studies reported that in addition to the expertise, the relationship between the patch author and the reviewer often affects the decision of whether to accept a review request [7, 24]. To capture this relationship, FPA counts the number of past reviews that a reviewer candidate had done for the patch author of a newly-submitted patch. Hence, the higher the FPA value is, the more the historical interaction between the reviewer candidate and the patch author. Then, it is more likely that the reviewer candidate will participate in the code review of the newly-submitted patch.

Review Participation Rate (RPR). RPR measures the extent to which a candidate participated in code review in the past. More specifically, RPR measures the proportion of past reviews that a reviewer candidate had participated compared to the number of past reviews to which the reviewer candidate was requested. A recent work of Ruangwan *et al.* [24] showed that RPR is one of the most influential factors affecting the participation decision of reviewers. The higher the RPR value is, the more the active reviewer candidate is; and the more likely that the reviewer will participate in a code review of a newly-submited patch.

Remaining Reviews (RR). A survey study reported that *"too many review requests"* is one of the reason that reviewers did not respond to review requests [24]. Hence, we use RR to represent the review workload of a reviewer candidate. The quantitative analysis of Ruangwan *et al.*showed that RR is one of the most influential factors affecting the participation decision of reviewers [24]. In addition, Baysal *et al.* [4] showed that the length of review queue, i.e., the number of pending review requests can have an impact on the code review timeliness. Hence, to measure RR, we count the number of review requests that a reviewer candidate received, yet the reviewer candidate did not participate at the time when the newly-submitted patch was created.

3.3 Generate Optimal Solutions

In this section, we describe the solution representation, the fitness functions for our two objectives, and the evolutionary search approach to generate optimal solutions (i.e., sets of reviewers).

3.3.1 Solution representation. We use a bit string to represent a solution candidate (i.e. a set of reviewer candidates). The length of a bit string is the number of all reviewer candidates. Each bit in the string has the value of 0 or 1. A bit value of 1 indicates that the corresponding reviewer is selected, while 0 indicates that the reviewer is excluded from a recommendation. For example, in a software project, there are five reviewer candidates (i.e., R_1, R_2, R_3, R_4, R_5). Then, in a solution candidate S, reviewer candidates R_1, R_2, and R_5 are selected for a recommendation. The solution candidate S can be represented in a bit string of `11001`.

The reviewer candidates in our approach are those who have participated in code reviews in the past. However, due to a large number of developers participated in code reviews, the length of a solution candidate can be long resulting in excess computation time. Hence, we shorten our candidate solutions by removing reviewer candidates who have at least three metrics with zero values. This is because the more metrics with zero values indicates the lower signal that the reviewer candidate will accept a review request. Note that we have experimented all possible metric thresholds for removing reviewer candidates (i.e., $t = \{1, 2, 3, 4, 5\}$). We found that removing reviewer candidates who have at least three metrics with zero values ($t = 3$) provides a reasonable length of candidate solutions (i.e., a median length of 23 - 381 reviewer candidates for a candidate solution), while it has a minimal impact on the accuracy of recommendation.

3.3.2 Fitness Functions. We now describe the calculation of the fitness functions for our two objectives.

Maximizing the chance of participating a review (CPR). For our first objective, we aim to find reviewers with maximum code ownership (CO), reviewing experience (RE), review participation rate (RPR) and those who are highly familiar with the patch author (FPA). In other words, the recommended reviewers of our approach are those who have related expertise, actively participated in code reviews in the past and reviewed many past patches for the patch author of a newly-submitted patch. To consider these four factors when recommending reviewers, we formulate the following fitness function:

$$CPR(S_i, p) = \sum_{r \in R} S_i(r)\Big(\alpha_1 CO(r,p) + \alpha_2 RE(r,p) + \alpha_3 FPA(r,p) + \alpha_4 RPR(r,p)\Big) \quad (3)$$

where p is a newly-submitted patch for a review, S_i is a candidate solution (i.e., a bit string of 0 or 1 for selecting reviewers), R is a set of all reviewer candidates. Each factor is weighted by the alpha (α) value where $\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 1$. Then, the higher the value of $CPR(S_i, p)$ is, the better the solution S_i (i.e., the set of selected reviewers) for a newly-submitted patch p is.

Minimizing the skewness of the reviewing workload distribution (SRW). To ensure that our recommendations will not burden a particular group of reviewers, we aim to balance reviewing workload among reviewers. In other words, the number of remaining reviews should not be skewed to a particular group of reviewers. Hence, we set an objective to minimize the skewness of the review workload distribution among reviewers. To do so, we adapt the calculation of Shannon's entropy [27] to measure the skewness of the remaining reviews (RR) distribution among reviewers. This is similar to the work of Hassan [11] who used Shanon's entropy to measure the distribution of modified code across modified files. More specifically, given a newly-submitted patch p and a solution candidate S_i, we formulate the following fitness function:

$$SRW(S_i, p) = \frac{1}{log_2|R|} \sum_{r \in R} (H(r) \times log_2 H(r)) \qquad H(r) = \frac{RR'(r)}{\sum_{k \in R} RR'(k)} \quad (4)$$

where R is a set of all reviewer candidates, $H(r)$ is a proportion of remaining reviews of a reviewer candidate r, and RR' is the number of remaining reviews (RR) including the newly-submitted patch if the reviewer r is selected in the solution S_i.

For example, reviewer candidates are R_1, R_2, R_3 and their remaining reviews are 3, 5, and 2, respectively. Given that a solution S_i selects R_1 and R_3 as recommended reviewers for a newly-submitted patch p, the RR' values will be 4 (=3+1), 5, and 3 (=2+1) for R_1, R_2, R_3 respectively. Then, the SRW for the solution S_i for the patch p is -0.98 ($= \frac{1}{log_2 3}(\frac{4}{12} log_2 \frac{4}{12} + \frac{5}{12} log_2 \frac{5}{12} + \frac{3}{12} log_2 \frac{3}{12})$). The lower the SRW value is, the better the spread of workload among reviewers.

3.3.3 Evolutionary search. We employ a multi-objective meta-heuristic algorithm, namely non-dominated sorting genetic algorithm (NSGA-II) [9] to search for solutions that meet the above two objectives. Algorithm 1 provides a pseudo-code of the NSGA-II algorithm. NSGA-II starts with randomly generating an initial population P_0 (i.e., a set of solution candidates). Then, the fitness of each solution candidate in the population P_0 is measured with respect to the two fitness functions described in Section 3.3.2. The initial population P_0 is then evolved to a new generation of solution candidates (i.e., an offspring population Q_0) through the selection and genetic operators, i.e., crossover and mutation. The selection operator ensures that the selection of solution candidates in the current population are proportional to their fitness values. The crossover operator takes two selected solution candidates as parents and swapped their bit strings to generate an offspring solution. The mutation operator randomly chooses certain bits in the string, and inverts the bits values.

At each generation t, the current population P_t and its offspring population Q_t is merged into a new population R_t. Then, NSGA-II sorts the solution candidates in the population R_t using the fast non-dominated sorting technique. This technique compares each solution with other solutions in the population R_t to find which solution dominates and which solution does not dominate other solutions. A solution S_1 is said to dominate another solution S_2 if S_1 is no worse than S_2 in all objectives and S_1 is strictly better than S_2 in at least one objective. After sorting, the fast non-dominated sorting technique provides Pareto fronts (i.e., sets of Pareto optimal solutions that are not dominated by any other solutions). For each of the Pereto front, NSGA-II calculates the crowding distance which is the sum of the distance in terms of fitness values between each solution and its nearest neighbours in the same front. Then, the Pareto fronts are sorted in the ascending order based on the crowding distance values. The population for a next generation P_{t+1} contains the first N solutions in the sorted Pareto fronts. The offspring population Q_{t+1} is then generated based on the population P_{t+1} through the selection and genetic operators. This evolution process is then repeated until a fixed number of generations has been reached. In the final generation, NSGA-II returns a set of Pareto optimal solutions.

Algorithm 1 NSGA-II pseudo-code [9]

```
P_0 ← Initial population is randomly generated with population size N
Evaluate Against Objective Functions
Selection Process ← Crossover and Mutation
Q_0 ← Create an offspring population
t = 0
while the number of generations not reached do
    R_t ← Merge(P_t + Q_t)
    F ← fast-non-dominated-sort (R_t)
    P_{t+1} = φ and j = 1
    while |P_{t+1}|+|F_j| ≤ N do
        Calculate crowding-distance-assignment(F_j)
        P_{t+1} ← P_{t+1} ∪ F_j
        j = j + 1
    end while
    Sort (F_j, ≺ n)
    P_{t+1} ← P_{t+1} ∪ F_j [1 : (N − |P_{t+1})]
    Q_{t+1} ← generated new population P_{t+1}
end while
```

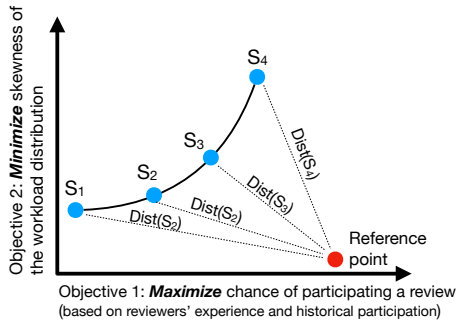

Figure 3: An illustrative example of identifying a knee point from the Pareto optimal solutions

3.4 Select a Solution

From the previous step, NSGA-II returns a set of Pareto optimal solutions, i.e., the sets of reviewers that meet the optimal trade-off of our two objectives. This set of solutions can be presented to the users for them to select. In cases where no explicit user preferences are provided, the so-called knee point approach is applied to select the most preferred solution among non-dominated solutions. This knee point approach has been widely used in the evolutionary search literature.

The knee point approach measures the Euclidean distance of each solution on the Pareto front from the reference point. Given the reference point is the maximum chance of participating a review (CPR_{max}) and the minimum skewness of the reviewing workload distribution (SRW_{min}), the Euclidean distance of a solution S_i is calculated as follows:

$$\text{Dist}(S_i) = \sqrt{(CPR_{max} - CPR(S_i))^2 + (SRW_{min} - SRW(S_i))^2} \quad (5)$$

The selected solution is the one closest to the reference point. Figure 3 provides an illustrative example for the knee point approach. Given that the Pareto optimal solutions returned by NSGA-II are S_1, S_2, S_3, and S_4 and their fitness values are shown in the plot, solution S_3 is closest to the reference point, i.e., the highest chance that the selected reviewers will accept a review request, while the reviewing workload is well distributed (low skewness). Hence, the solution S_3 will be selected, and the reviewers associated with this solution will be recommended.

4 EXPERIMENTAL DESIGN

4.1 Research Questions

To evaluate our WLRRec, we formulate the following research questions.

(RQ 1) Can our WRLRec recommend reviewers who are potentially suitable for a newly-submitted patch?
We set out this RQ for a sanity check to determine whether our approach can recommend *actual* reviewers for a newly-submitted patch. In addition, we compare our approach with the Random Search optimization technique [13] which has been a common baseline for most of the search based metaheuristic algorithms.

Table 1: An overview of the evaluation datasets

Project	Period	# Patches	#Reviewers
Android	10/2008 - 12/2014	36,771	2,049
Qt	5/2011 - 12/2014	65,815	1,238
OpenStack	7/2011 - 12/2014	108,788	3,734
LibreOffice	3/2012 - 11/2016	18,716	410

(RQ 2) Does our WRLRec benefit from the multi-objective search-based approach?
Since our WLRRec considers two objectives, we set out this RQ to empirically evaluate how well the multi-objective approach compared to the single-objective approach. To answer this RQ, we implemented the traditional single-objective genetic algorithm (GA) using either CPR or SRW as an objective. We then compare the performance of our WLRRec against these two single-objective approaches, i.e., GA-CPR and GA-SRW. Indeed, the GA-CPR approach is closely similar to RevRec of Ouli et al.[19], who used the genetic algorithm (GA) and considered the reviewing experience and historical interaction.

(RQ 3) Does the choice of search algorithms impact the performance of our WRLRec?
Our WLRRec leverages the multi-objective optimization algorithm to recommend reviewers. Aside from NSGA-II, there are other multi-objective optimization algorithms. Hence, we set out this RQ to evaluate our WLRRec when using two recently developed multi-objective evolutionary algorithms: Multiobjective Cellular Genetic Algorithm (MOCell) [18] and the Strength-based Evolutionary Algorithm (SPEA2) [39].

4.2 Datasets

In this work, we use a code review dataset of four large open source software projects that actively use modern code reviews, i.e., Android, Qt, OpenStack, and LibreOffice. These projects have a large number of the patches recorded in the code review tool. For Android, Qt, and OpenStack, we use the review datasets of Hamasaki et al. [10] which was often used in prior studies [19, 24, 32]. For LibreOffice, we use the review dataset of Yang et al. [35]. The datasets include patch information, review discussion, and developer information. Below, we describe our data preparation approach, which consist of three main steps.

(Step 1) Cleaning Datasets. In this paper, we use the patches that were marked as either *merged* or *abandoned*. In addition, we exclude the patches that are self-reviewed (i.e., only the author of the patch was a reviewer) and the patches related to the version control system activities (e.g., branch-merging patches). Note that we search for a keyword "merge branch" or "merge" in the description of the patch to identify the branch-merging patches. These patches are excluded from our evaluation because there might be no reviewers who actually review those patches. Since the code review tools of the studied projects are tightly integrated with automated checking systems (e.g., Continuous Integration test system), we remove the accounts of automated checking systems (e.g., Jenkins CI or sanity checks) in the data sets. Similar to prior work [24], we use

a semi-automated approach to identify the accounts of automated checking systems. Finally, we use the approach of Bird et al. [5] to identify and merge alias emails of developers. In total, we obtain 230,090 patches with 7,431 reviewers that spread across four open source projects. Table 1 provides an overview of our datasets.

(Step 2) Spliting Datasets. To evaluate our approach, we split each dataset into two sub-datasets: (1) the most recent 10% of the patches and (2) the remaining 90% of the patches. The 10% sub-dataset is used to evaluate our approach, i.e., the patches in this sub-dataset are considered as newly-submitted patches. The 90% sub-dataset is used for building a pool of reviewer candidates and computing reviewer metrics. As described in Section 3.3.1, reviewer candidates are those (1) who have provided either a vote score or a comment to at least one patch in the 90% sub-dataset and (2) whose at least three reviewer metrics have a value greater than zero.

(Step 3) Generating Ground-Truth Data. For the ground-truth data of the patches in the 10% sub-dataset, we identify two groups of reviewers: (1) actual reviewers and (2) potential reviewers. Given a patch p in the 10% sub-dataset, the ***actual reviewers*** of the patch p are those who actually reviewed the patch p by either providing a vote score or a comment. These actual reviewers are typically used to evaluate the reviewer recommendation approaches [19, 32, 38]. Although these reviewers have actually reviewed in historical data, there is no guarantee that these reviewers are the only group of suitable reviewers. Moreover, a recent work of Kovalenko et al. [15] has shown that although a reviewer recommendation approach achieves a good accuracy when the evaluation is based on historical data, developers may not always select the recommended reviewers when the approach is deployed due to several factors, e.g., reviewers' availability.

Hence, to evaluate our approach, we expand our ground-truth data to include potential reviewers. ***Potential reviewers*** are the reviewers who should be able to review the changed files if they have done a review for some of these files in other patches. Specifically, given a patch p, the potential reviewers of the patch p are those who were the actual reviewers of the other patches in the 10% sub-dataset that made changes to at least one file, and this file is one of the changed files of the patch p. For example, a patch p_1 made changes to files A and B, while a patch p_2 made changes to files B and C. Reviewers R_1 and R_2 are the actual reviewers of the patch p_1, while reviewers R_3 and R_4 are the actual reviewers of the patch p_2. Hence, the potential reviewers for the patch p_1 are R_3 and R_4 as they reviewed file B (i.e., the common changed file in patches p_1 and p_2).

4.3 Evaluation Analysis

To evaluate our approach, we use four performance measures. Then, we perform a statistical analysis to determine the statistical difference between our approach and the other approaches.

4.3.1 Performance measures. We evaluate our WLRRec based on two perspectives. First, we evaluate our approach from the perspective of recommendation systems. Hence, for each newly-submitted patch (i.e., a patch in the 10% sub-dataset), we use Precision, Recall, and F-measure to measure the accuracy of our approach. Second, we evaluate our approach from the perspective of search-based software engineering. This measure has been used in previous work (e.g. [20]) as a performance indicator for multi-objective optimization. Hence, for each newly-submitted patch, we use Hypervolume to evaluate the performance of search algorithms and to guide the search. Below, we describe the calculation of each performance measure:

Precision measures the proportion of the recommended reviewers that are in the ground-truth data. We measure precision for a newly-submitted patch p_i using a calculation of $P(p_i) = \frac{|\text{rec}(p_i) \cap \text{g}(p_i)|}{|\text{rec}(p_i)|}$, where $\text{rec}(p_i)$ is a set of recommended reviewers and $\text{g}(p_i)$ is a set of reviewers in the ground-truth data .

Recall measures the proportion of reviewers in the ground-truth data that are recommended by the approach. We measure recall for a newly-submitted patch p_i using a calculation of $R(p_i) = \frac{|\text{rec}(p_i) \cap \text{g}(p_i)|}{|\text{g}(p_i)|}$, where $\text{rec}(p_i)$ is a set of recommended reviewers and $\text{g}(p_i)$ is a set of reviewers in the ground-truth data .

F-measure is the harmonic mean of the precision and recall using a calculation of $F(p_i) = \frac{2(P(p_i) \times R(p_i))}{P(p_i)+R(p_i)}$

Hypervolume is a quality indicator for the volume of the space covered by the non-dominated solutions from the search algorithm [40]. It indicates the convergence and diversity of the solutions on a Pareto front (e.g. the higher hypervolume, the better performance), which using the following calculation [26]: $HV = volume\left(\bigcup_{i=1}^{S} v_i\right)$ where S is a set of solutions from the Pareto front and v_i is the hypercube space established between each solution i and distance (reference) point by all solutions.

4.3.2 Statistical Analysis. To compare the performance between our WLRRec and other benchmarks, we compute the performance gain of WLRRec (where pm indicates to precision, recall, F-score, or MRR) over another compared the benchmark Y using the following calculation: $Gain(pm, Y) = \frac{WLRRec_{pm} - Y_{pm}}{Y_{pm}} \times 100.\%$

In addition, we use the Wilcoxon Signed Rank test [8] ($\alpha = 0.05$) to determine whether the performance of our WLRRec is statistically better than the other approaches. The Wilcoxon Signed Rank test is a paired-wise non-parametric test which does not assume a normal distribution. In addition, we measure the effect size (i.e., the magnitude of difference) using the Vargha and Delaney's $\hat{A}_{XY}$ non-parametric effect size measure [33]. The $\hat{A}_{XY}$ measures the probability that the performance achieved from the approach X is better than the performance achieved by the approach Y. Considering a performance measure pm (i.e., precision, recall, f-measure, and Hypervolume), the effect size is measured using the following calculation: $\hat{A}_{XY}(pm) = \frac{[\#(X_{pm} > Y_{pm}) + (0.5 * \#(X_{pm} = Y_{pm}))]}{P}$, where X_{pm} is the number of patches where the performance pm of the approach X (i.e., our approach) and Y_{pm} is the performance pm of the approach Y (e.g., random search), and P is the number of newly-submitted patches (i.e., the size of the 10% sub-dataset). The difference is considered as trivial for $\hat{A}_{XY} \leq 0.147$, small for $0.147 < \hat{A}_{XY} \leq 0.33$, medium for $0.33 < \hat{A}_{XY} \leq 0.474$, and large for $\hat{A}_{XY} > 0.474$ [12].

4.4 Experimental Settings

Our approach was implemented in the MOEA Framework.[1] We employed tournament selection method and set the size of the initial population to 100. The number of generations was set to 100, 000. Crossover probability was set to 0.9, mutation probability was 0.1, and reproduction probability was 0.2. We set the parameters $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ to 0.25 as default parameters.

5 RESULTS

(RQ1) Can our WRLRec recommend reviewers who are potentially suitable for a newly-submitted patch?

Results. For a sanity check, the first row of Table 2 presents the performance results of our WLRRec when using only actual reviewers as a ground-truth. We find that our WLRRec approach achieves a precision of 16%-20%, a recall of 30%-37%, an F-measure of 17%-28%, and a hypervolume of 72%-83%. These results suggest that our WLRRec can identify a reviewer who actually reviewed the patches in the past. However, as discussed in Section 4.2 (Step-3), there is no guarantee that these actual reviewers are the only group of suitable reviewers. Hence, the goal of our work is not limited to find the exact group of actual reviewers, but to find potential reviewers who might be able to review the patch in the future. For the remaining results, we evaluate our WLRRec when using the combination of actual and potential reviewers as ground-truths.

Our WLRRec can recommend reviewers who are likely to accept a review request with an F-measure of 0.32 - 0.43, which is 137%-260% better than the random search approach. Tables 2 and 3 show that our WLRRec can recommend reviewers who are likely to accept a review request with a precision value of 31% for Android, 34% for LibreOffice, 36% for QT, and 32% for OpenStack. Furthermore, Table 2 shows that our WLRRec achieves a recall value of 50% for Android, LibreOffice and QT, and 53% for OpenStack. Table 3 also shows that our WLRRec achieves a recall value 201%-277% better than the random search approach. The hypervolume values of 72%-84% achieved by our WLRRec also indicate that our multi-objective search algorithm is 85%-214% better than a random search approach. The Wilcoxon Signed Rank tests ($p < 0.001$) and the large $\hat{A}_{XY}$ effect size values ($\hat{A}_{XY} > 0.474$) confirm that the performance of our WLRRec is statistically better than that of the random search approach in terms of precision, recall, F-measure, and hypervolume.

Discussion. The results of RQ1 suggest that our WLRRec can recommend reviewers who are potentially suitable for a newly-submitted patch. More specifically, we find that when considering 4+1 key reviewer metrics, our WLRRec can recommend reviewers who actually reviewed the patches in the past. In addition, when expanding the ground-truth data to include potential reviewers (e.g., those who should be able to review the newly-submitted patch), about half of the potential reviewers can be identified by our WLRRec (c.f. the recall values of our approach). These empirical results suggest the effectiveness of considering different factors when recommending reviewers for a newly-submitted patch.

[1] http://moeaframework.org/index.html

(RQ2) Does our WRLRec benefit from the multi-objective search-based approach?

Results. Tables 2 shows that the single-objective approach that maximizes the chance of participating a review (GA-CPR) achieves a precision of 15%-17%, a recall of 18%-25%, and an F-measure of 17%-21%. On the other hand, the single-objective approach that minimizes the skewness of the reviewing workload distribution (GA-SRW) achieves a precision of 17%-20%, a recall of 21%-27%, and an F-measure of 18%-22%. Note that we did not measure hypervolume for the GA-CPR and GA-SRW approaches since this performance measure is not applicable for the single-objective approaches.

Our WLRRec outperforms the single-objective approaches with the performance gain of 55%-142% for precision and 78%-178% for recall. Table 3 shows that our WLRRec achieves 88%-142% higher precision, 111%-178 higher recall, 52%-124% higher F-measure than the GA-CPR approach. Similarly, we also find that our WLRRec achieves 55%-101% higher precision, 96%-138% higher recall, 45%-111% higher F-measure than the GA-SRW approach. The Wilcoxon Singed Rank tests and the magnitude of the differences measured by the $\hat{A}_{XY}$ effect size also confirm that the difference is statistically significant with a large magnitude of difference ($\hat{A}_{XY} > 0.474$).

Discussion. The results of our RQ2 indicate that our WLRRec which uses a multi-objective approach (i.e., maximizing the chance of participating a review, while minimizing the skewness of the reviewing workload distribution) is statistically better than the single-objective approaches when recommending reviewers who are potentially suitable for a newly-submitted patch. These results suggest that considering multiple objectives at the same time would allow us to find other potential reviewers that might be overlooked in the previous approaches [19, 32]. Furthermore, Table 2 shows that the GA-SRW approach (which considers only the workload distribution) achieves a recall relatively better than the GA-CPR approach which only focuses on the reviewing experience, historical interaction, and activeness of reviewers (similar to RevRec [19]). These empirical results highlight the benefits of using the multi-objective search-based approaches and considering the workload distribution of reviewers when recommending reviewers for a newly-submitted patch.

(RQ3) Does the choice of search algorithms impact the performance of our WRLRec?

Results. Tables 2 shows that when using MOCell instead of NSGA-II to search for the optimal solutions, this approach achieves a precision of 23%-36%, a recall of 26%-36%, an F-measure of 22%-24%, and a hypervolume of 56%-67%. Similarly, using SPEA2 to search for the optimal solutions achieves a precision of 21%-26%, a recall of 32%-40%, an F-measure of 22%-29%, and a hypervolume of 52%-58%.

Our WLRRec with NSGA-II achieves 31%-95% and 19%-95% higher F-measure than the MOCell and SPEA2 approaches, respectively. Table 3 shows that our WLRRec which uses NSGA-II achieves 31%-48% higher precision, 45%-95% higher recall, 31%-95% higher F-measure, and 21%-31% higher hypervolume than the MOCell approach. Similarly, we also find that our WLRRec achieves 0%-52% higher precision, 26%-59% higher recall, 19%-95% higher

Table 2: The precision, recall, f-measure, and hypervolumn values ([0,1]) of our WLRRec. The first row presents the results when using the actual reviewers as a ground-truth, while the other rows present the results when using the combinations of actual and potential reviewers as a ground-truth.

		Android				LibreOffice				QT				OpenStack			
GT	**Techniques**	**P**	**R**	**F1**	**HV**	**P**	**R**	**F1**	**HV**	**P**	**R**	**F1**	**HV**	**P**	**R**	**F1**	**HV**
Act.	WLRRec	0.20	0.30	0.28	0.72	0.16	0.31	0.17	0.74	0.19	0.37	0.27	0.83	0.20	0.34	0.22	0.82
Act. + Pot. Rev.	WLRRec	0.31	0.50	0.35	0.72	0.34	0.50	0.43	0.74	0.36	0.50	0.38	0.84	0.32	0.53	0.32	0.82
	(RQ1) Random	0.08	0.16	0.10	0.35	0.12	0.17	0.16	0.40	0.12	0.15	0.13	0.27	0.10	0.14	0.14	0.28
	(RQ2) GA-CPR	0.15	0.18	0.17	-	0.16	0.23	0.19	-	0.15	0.19	0.17	-	0.17	0.25	0.21	-
	(RQ2) GA-SRW	0.20	0.21	0.18	-	0.17	0.24	0.21	-	0.20	0.21	0.18	-	0.18	0.27	0.22	-
	(RQ3) MOCell	0.24	0.26	0.24	0.56	0.23	0.30	0.22	0.61	0.36	0.34	0.29	0.67	0.23	0.36	0.22	0.63
	(RQ3) SPEA2	0.27	0.40	0.29	0.52	0.26	0.32	0.22	0.58	0.36	0.34	0.29	0.57	0.21	0.34	0.27	0.57

Table 3: The performance gain of our proposed WLRRec over the other benchmarks.

Techniques	Android				LibreOffice				QT				OpenStack			
	P	**R**	**F1**	**HV**	**P**	**R**	**F1**	**HV**	**P**	**R**	**F1**	**HV**	**P**	**R**	**F1**	**HV**
WLRRec–Random	288%	207%	260%	108%	175%	201%	161%	85%	203%	233%	189%	214%	217%	277%	137%	199%
WLRRec–GA-CPR	107%	178%	104%	-	113%	117%	124%	-	142%	163%	123%	-	88%	111%	52%	-
WLRRec–GA-SRW	55%	138%	92%	-	101%	108%	103%	-	82%	138%	111%	-	78%	96%	45%	-
WLRRec–MOCell	31%	95%	42%	28%	48%	68%	95%	21%	0%	48%	31%	25%	42%	45%	45%	31%
WLRRec–SPEA2	17%	26%	19%	37%	30%	59%	95%	29%	0%	48%	31%	47%	52%	55%	19%	43%

F-measure, and 29%-47% higher hypervolume than the SPEA2 approach. The Wilcoxon Singed Rank tests and the magnitude of the differences measured by the $\hat{A}_{XY}$ effect size also show that the difference is statistically significant with a large magnitude of difference ($\hat{A}_{XY} > 0.474$).

Discussion. The results of our RQ3 indicate that the choice of the multi-objective search-based algorithms has an impact on the performance of our approach. More specifically, when using the other multi-objective algorithms (i.e., MOCell and SPEA2), the performance of our approach decreases in terms of recall, f-measure, and hypervolume. In addition, the higher hypervolume value of our WLRRec indicates that our approach finds the solutions that satisfy the two objectives better that other two multi-objective approaches. These empirical results suggest that the NSGA-II algorithm that we leveraged is an appropriate multi-objective approach to find solutions in this problem domain.

6 THREATS TO VALIDITY

Construct threat to validity is related to the ground-truth set of reviewers for evaluating our approach. While the actual reviewers are recorded in historical data, there is no guarantee that these actual reviewers are the only group of suitable reviewers. Ouni *et al.* [19] point out that potential reviewers may be assigned to a code change that such reviewers do not contribute to it. This is due to several reasons including the current workload, the availability, and the social relationship with the patch author. To mitigate this threat, as suggested by Ouni *et al.* [19], we considered the ground truth as the set of potential reviewers who should be able to review the changed files if they have done a review for some of these files in other patches, which may be more realistic for evaluation than the set of actual reviewers. Moreover, the goal of our work is not limited to find the exact group of actual reviewers, but to find potential reviewers who might be able to review the patch in the future.

External threats to validity is related to the generalizability of our results. Although we empirically evaluated our approach on four large size open-source systems from different application domains, Android, Qt, OpenStack, and LibreOffice, we do not claim that the same results would be achieved with other projects or other periods of time.

7 CONCLUSION AND FUTURE WORK

In this paper, we develop a multi-objective search-based approach called Workload-aware Reviewer Recommendation (**WLRRec**) to find reviewers for a newly-submitted patch. Our results suggest that: (1) when considering five reviewer metrics, our WLRRec can recommend reviewers who are potentially suitable for a newly-submitted patch with 19% - 260% higher F-measure than the five benchmarks; (2) including an objective to minimize the skewness of the review workload distribution would be beneficial to find other potential reviewers that might be overlooked by the other approaches that focus on reviewing experience; and (3) the multi-objective meta-heuristic algorithm, NSGA-II, can be used to search for reviewers who are potentially suitable for a newly-submitted patch. Our empirical results shed the light on the potential of using a wider range of information and leveraging the multi-objective meta-heuristic algorithm to find reviewers who are potentially suitable for a newly-submitted patch.

Our future work will incorporate other factors which may affect the quality and productivity of the code review process, leading to the formulation of new objectives and constraints which should be considered in the search for generating optimal solutions. Currently, our work considers only one newly-submitted code patch at a time. Thus, our future work will extend the consideration of all code patches that are currently sitting in the queue at the same time. This may require a new solution representation.

Acknowledgement. C. Tantithamthavorn was partially supported by ARC DECRA funding scheme (DE200100941).

8 REFERENCES

[1] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 international conference on software engineering*. IEEE Press, 712–721.

[2] Vipin Balachandran. 2013. Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 931–940.

[3] Gabriele Bavota and Barbara Russo. 2015. Four eyes are better than two: On the impact of code reviews on software quality. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 81–90.

[4] Olga Baysal, Oleksii Kononenko, Reid Holmes, and Michael W. Godfrey. 2015. Investigating Technical and Non-Technical Factors Influencing Modern Code Review. *Journal of Empirical Software Engineering (EMSE)* 21, 3 (2015), 932–959.

[5] Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. 2006. Mining email social networks. In *Proceedings of the 2006 international workshop on Mining software repositories*. 137–143.

[6] Christian Bird, Nachiappan Nagappan, Brendan Murphy, Harald Gall, and Premkumar Devanbu. 2011. Don't touch my code!: examining the effects of ownership on software quality. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, 4–14.

[7] Amiangshu Bosu, Jeffrey C. Carver, Christian Bird, Jonathan Orbeck, and Christopher Chockley. 2017. Process Aspects and Social Dynamics of Contemporary Code Review: Insights from Open Source Development and Industrial Practice at Microsoft. *Transactions on Software Engineering (TSE)* 43, 1 (2017), 56–75.

[8] J Cohen. 1988. Statistical power analysis for the behavioral sciences. 1988, Hillsdale, NJ: L. *Lawrence Earlbaum Associates* 2 (1988).

[9] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197.

[10] Kazuki Hamasaki, Raula Gaikovina Kula, Norihiro Yoshida, AE Cruz, Kenji Fujiwara, and Hajimu Iida. 2013. Who does what during a code review? datasets of oss peer review repositories. In *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 49–52.

[11] Ahmed E Hassan. 2009. Predicting faults using the complexity of code changes. In *Proceedings of the 31st International Conference on Software Engineering*. IEEE, 78–88.

[12] Melinda R Hess and Jeffrey D Kromrey. 2004. Robust confidence intervals for effect sizes: A comparative study of Cohen'sd and Cliff's delta under non-normality and heterogeneous variances. In *annual meeting of the American Educational Research Association*. 1–30.

[13] Dean C Karnopp. 1963. Random search techniques for optimization problems. *Automatica* 1, 2-3 (1963), 111–121.

[14] Oleksii Kononenko, Olga Baysal, Latifa Guerrouj, Yaxin Cao, and Michael W Godfrey. 2015. Investigating code review quality: Do people and participation matter?. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 111–120.

[15] Vladimir Kovalenko, Nava Tintarev, Evgeny Pasynkov, Christian Bird, and Alberto Bacchelli. 2018. Does reviewer recommendation help developers? *IEEE Transactions on Software Engineering* to appear (2018), 1–23.

[16] Laura MacLeod, Michaela Greiler, Margaret-Anne Storey, Christian Bird, and Jacek Czerwonka. 2018. Code Reviewing in the Trenches. *IEEE Software* 35 (2018), 34–42.

[17] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. 2016. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering* 21, 5 (2016), 2146–2189.

[18] Antonio J Nebro, Juan J Durillo, Francisco Luna, Bernabé Dorronsoro, and Enrique Alba. 2009. MOCell: A cellular genetic algorithm for multiobjective optimization. *International Journal of Intelligent Systems* 24, 7 (2009), 726–746.

[19] Ali Ouni, Raula Gaikovina Kula, and Katsuro Inoue. 2016. Search-based peer reviewers recommendation in modern code review. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 367–377.

[20] Ali Ouni, Raula Gaikovina Kula, Marouane Kessentini, Takashi Ishio, Daniel M German, and Katsuro Inoue. 2017. Search-based software library recommendation using multi-objective optimization. *Information and Software Technology* 83 (2017), 55–75.

[21] Mohammad Masudur Rahman, Chanchal K Roy, and Jason A Collins. 2016. Correct: code reviewer recommendation in github based on cross-project and technology experience. In *Proceedings of the 38th International Conference on Software Engineering Companion*. 222–231.

[22] Peter C Rigby and Christian Bird. 2013. Convergent contemporary software peer review practices. In *Proceedings of the Joint Meeting on Foundations of Software Engineering (FSE)*. ACM, 202–212.

[23] Peter C Rigby and Margaret-Anne Storey. 2011. Understanding broadcast based peer review on open source software projects. In *Proceedings of the International Conference on Software Engineering*. ACM, 541–550.

[24] Shade Ruangwan, Patanamon Thongtanunam, Akinori Ihara, and Kenichi Matsumoto. 2019. The Impact of Human Factors on the Participation Decision of Reviewers in Modern Code Review. *Journal of Empirical Software Engineering (EMSE)* (2019).

[25] Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern Code Review: A Case Study at Google. In *Companion Proceedings of the International Conference on Software Engineering (ICSE)*. 181–190.

[26] Raphael Saraiva, Allysson Allex Araujo, Altino Dantas, Italo Yeltsin, and Jerffeson Souza. 2017. Incorporating decision maker's preferences in a multi-objective approach for the software release planning. *Journal of the Brazilian Computer Society* 23, 1 (2017), 11.

[27] Claude Elwood Shannon. 1948. A mathematical theory of communication. *Bell system technical journal* 27, 3 (1948), 379–423.

[28] Patanamon Thongtanunam, Shane McIntosh, Ahmed E Hassan, and Hajimu Iida. 2015. Investigating Code Review Practices in Defective Files: An Empirical Study of the Qt System. In *Proceedings of the 12th International Working Conference on Mining Software Repositories (MSR)*. 168–179.

[29] Patanamon Thongtanunam, Shane McIntosh, Ahmed E. Hassan, and Hajimu Iida. 2016. Revisiting Code Ownership and its Relationship with Software Quality in the Scope of Modern Code Review. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 1039–1050.

[30] Patanamon Thongtanunam, Shane McIntosh, Ahmed E Hassan, and Hajimu Iida. 2016. Revisiting code ownership and its relationship with software quality in the scope of modern code review. In *Proceedings of the 38th international conference on software engineering*. ACM, 1039–1050.

[31] Patanamon Thongtanunam, Shane McIntosh, Ahmed E Hassan, and Hajimu Iida. 2017. Review participation in modern code review. *Empirical Software Engineering* 22, 2 (2017), 768–817.

[32] 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 *Proceedings of the International Conference on Software Analysis, Evolution and Re-engineering (SANER)*. IEEE, 141–150.

[33] András Vargha and Harold D Delaney. 2000. A critique and improvement of the CL common language effect size statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (2000), 101–132.

[34] 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 *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*. IEEE, 261–270.

[35] Xin Yang, Raula Gaikovina Kula, Norihiro Yoshida, and Hajimu Iida. 2016. Mining the modern code review repositories: A dataset of people, process and product. In *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 460–463.

[36] Yue Yu, Huaimin Wang, Gang Yin, and Charles X Ling. 2014. Reviewer recommender of pull-requests in GitHub. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 609–612.

[37] 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.

[38] 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.

[39] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. 2001. SPEA2: Improving the strength Pareto evolutionary algorithm. *TIK-report* 103 (2001).

[40] Eckart Zitzler and Lothar Thiele. 1999. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE transactions on Evolutionary Computation* 3, 4 (1999), 257–271.