A FIRST LOOK AT FAIRNESS OF MACHINE LEARNING BASED CODE REVIEWER RECOMMENDATION

MOHAMMAD MAHDI MOHAJER

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE

GRADUATE PROGRAM IN ELECTRICAL ENGINEERING & COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO

APRIL 2024

Abstract

The fairness of machine learning (ML) approaches is critical to the reliability of modern artificial intelligence systems. Despite extensive study on this topic, the fairness of ML models in the software engineering (SE) domain has yet to be well explored. As a result, many ML-powered software systems, particularly those utilized in the software engineering community, continue to be prone to fairness issues. Taking one of the typical SE tasks, i.e., code reviewer recommendation, as a subject, this work conducts the first study toward investigating the fairness of ML applications in the SE domain, explicitly focusing on the code reviewer recommendation task. Our empirical study demonstrates that current state-of-the-art ML-based code reviewer recommendation techniques exhibit unfairness and discriminating behaviors. Specifically, male reviewers get, on average, 7.25% more recommendations than female code reviewers compared to their distribution in the reviewer set. This work also discusses why the studied ML-based code reviewer recommendation systems are unfair and provides solutions to mitigate the unfairness. For instance, these techniques may recommend male reviewers at a significantly higher rate than female reviewers in a discriminatory manner. Our study further indicates that the existing mitigation methods can enhance fairness

significantly in projects with a similar distribution of protected and privileged groups. Still, their effectiveness in improving fairness on imbalanced or skewed data is limited. Eventually, we suggest a solution to overcome the drawbacks of existing mitigation techniques and tackle bias in imbalanced or skewed datasets.

Acknowledgements

I would like to express my gratitude to my supervisor, **Prof. Song Wang**, and my co-supervisor, **Prof. Alvine Boaye Belle**, for their invaluable assistance in conceptualizing the main idea for my work, guiding me through the research obstacles, and providing financial support throughout the research process. I am also deeply grateful to **Prof. Laleh Seyyed-kalantari** and **Prof. Manar Jammal**, members of my thesis examining committee, for their time and valuable feedback on my thesis.

Table of Contents

List of Tables

List of Figures

Abbreviations

Table 1: Abbreviations List

Abbreviation	**Definition**
AI / ML	Artifical Intelligence / Machine Learning
SE	Software Engineering
SVM	Support Vector Machines
CR	Code Review
MCR	Modern Code Review
CRR	Code Reviewer Recommendation
SPD	Statistical Parity Difference
NDKL	Normalized Discounted Cumulative Kullback-Leiber divergence
MRR	Mean Reciprocal Rank
DG	DetGreedy Algorithm
DR	DetRelaxed Algorithm
IRR	Improved Re-Ranking Algorithm

Chapter 1

Introduction

1.1 Background and Context

Machine Learning (ML) approaches and models have increasingly been used to develop modern software [1] to assist developers in different tasks, e.g., defect prediction [2, 3], software bug triage [4], and code reviewer recommendation [5, 6]. Meanwhile, the broad adoption of ML has given rise to new concerns and issues regarding the trustworthiness and ethicality of such systems, one of which is the issue of fairness [7, 8]. In machine learning, fairness means ensuring that algorithms and models do not favor or discriminate against specific groups or people based on their human-related and demographic aspects, such as race, gender, age, or ethnicity [9, 8, 7, 10]. For example, judicial systems in the US that use machine learning for criminal predictions have been shown to exhibit bias against African-American individuals [7]. These systems predict a higher rate of criminality for African Americans in a discriminatory

manner [7]. Existing studies have shown that unfair machine learning applications can seriously impact on people's lives [11, 12, 13, 14, 15]. For instance, Amazon's ML-based hiring algorithm discriminates against female candidates [7]. The fairness problem has also been shown to exist in other human-sensitive areas using machine learning for making decisions, such as judicial evaluation systems, credit scoring, and loan application filtering [7, 11, 14, 15]. Additionally, previous research [16, 7, 17] has revealed that implicit biases and concerns regarding fairness across various aspects, including gender, geography, socioeconomic status, age, ethnicity, and disability, can significantly affect individuals' lives. These factors may also contribute to a diversity crisis, particularly within the software engineering community, resulting in a workforce and environment that are more exclusive than inclusive.

1.2 Problem Statement

Even though previous studies [17, 18, 19, 20, 21, 22, 23, 24, 25, 26] have extensively examined the fairness of ML applications, while most of these studies mainly focus on general ML applications, more is needed about the fairness of ML applications in the software engineering domain, e.g., automated bug triage [4]. For our research, we take one of the typical SE tasks, i.e., code reviewer recommendation, as a subject to explore the fairness of ML applications in the SE domain. Specifically, code reviewer recommendation systems are widely used in modern software development to identify the most appropriate code reviewers for a code change [27, 28]. Recently, many ML-based code reviewer recommendation systems have been proposed. For instance, Patanamon et al. [29] proposed RevFinder, which used similarity of

previously reviewed file path to recommend an appropriate code-reviewer, and Pandya et al. [6] proposed CORMS, which leveraged similarity analysis and support vector machine (SVM) models to recommend reviewers. Although these examined code reviewer recommendation approaches can perform well, none of the fairness characteristics (e.g., race, age, and gender) were considered when recommending reviewers. As a result, there may be fairness issues in such systems that have not previously been investigated, which can adversely impact reviewers' activities [25, 7, 26]. For instance, in such systems, the final recommendation list may exhibit a need for more representation of female reviewers, as they might be recommended less frequently than their male counterparts in a biased manner.

1.3 Objective

To address the aforementioned concerns, in this work, we look into the problem of assessing fairness issues in ML-based code reviewer recommendation systems for the first time in SE research. Specifically, we hypothesize that machine learning-based code reviewer recommendation systems may be unfair in their recommendations due to latent bias in the data utilized to train them. We empirically study two recent ML-based code reviewer recommendation systems, RevFinder [29] and CORMS [6]. To support and validate our findings, we used the same dataset from CORMS [6] to build these systems and run our experiments. Note that when exploring the fairness of ML-based code reviewer recommendation systems, we only consider the factor of gender. We do this mainly because collecting data to identify sensitive factors such as age or race is difficult, as reviewers and code review platforms often do not

disclose this information. Gender information is also often not specified by the reviewers. However, compared to other factors, this information can still be derived from other sources, e.g., social media accounts, personal websites, institutional websites, and GitHub profiles. Furthermore, as previously stated, gender emerges as the primary factor influencing the diversity crisis in software development [16]. Consequently, by examining fairness through gender as our studied sensitive attribute, we can foster diversity within software engineering communities and effectively address the challenges surrounding the diversity crisis. Moreover, since obtaining non-binary gender information is difficult, and in line with previous fairness research [17, 7, 21, 20, 8] that used gender as the sensitive attribute, we treat gender as a binary attribute in our analysis. Nevertheless, it is essential to emphasize that our study's scope is not confined to binary values and can be expanded to include non-binary genders if we acquire adequate data (details are provided in Sections 3.3 and 3.4).

1.4 Contributions

Our experimental results show that both RevFinder and CORMS have unfair behavior in their recommendations. Specifically, they favor male reviewers over female reviewers. For example, in the Node.js project, male reviewers recommended by CORMS have approximately 85% more chance of being recommended for new code review requests, which is 16% more than the fair condition (details are in Chapter 4). We further explore the underlying factors that contribute to the unfairness of ML-based code reviewer recommendation systems, e.g., popularity bias, and whether the existing unfairness mitigation approaches [30, 31, 32, 33] can help improve

the fairness of these systems. Our experiment results show that the existing mitigation approaches can improve fairness, but only sometimes across all projects. Thus, we propose a new mitigation method to counteract the limitations of existing unfairness mitigation strategies. Our proposed technique can enhance fairness in code reviewer recommendation systems in multiple circumstances. This study contributes to creating fairer ML-based code reviewer recommendation systems, reducing gender-based discrimination, and promoting societal equality and fairness. Those creating systems that engage with humans can apply the findings and methodology from our research to ensure their software products are equitable and contribute to equality among users. The methods used in our study also apply to other types of recommendation systems, such as team recommendation systems for collaborative software development [34].

In summary, this work makes the following contributions:

- We conduct an empirical study to investigate the fairness of two state-of-the-art code reviewer recommendation systems. To the best of our knowledge, this is the first study on fairness analysis of ML applications in the software engineering domain.

- We analyze the underlying factors that influence the outcomes of code reviewer recommendation systems and demonstrate that the existing unfairness mitigation methods can be utilized to alleviate the unfairness in ML-based code reviewer recommendation systems. The improvement can be as high as 100%, effectively removing bias from recommendations, particularly in projects where protected and privileged groups have a balanced distribution.

- We show that the current mitigation approaches have limitations in terms of fairness improvement for projects with imbalanced or skewed data. Motivated by that, we propose a new unfairness mitigation algorithm to solve the issues associated with the previous techniques. Our algorithm can significantly outperform the two examined baselines.

- We release our experiments' dataset and source code to help other researchers replicate and extend our study (details in Appendix B).

1.5 Significance of Research

This study contributes to creating fairer AI-based software, specifically, code reviewer recommendation systems, reducing gender-based discrimination, and promoting societal equality and fairness. Those creating systems that engage with humans can apply the findings and methodology from my research to ensure their software products are equitable and contribute to equality among users. The methods used in our study also apply to other types of recommendation systems, such as expert recommendation systems for software teams.

1.6 Thesis Organization

The rest of this work is organized as follows. Chapter 2 presents the literature review of this work. Chapters 3 and 4 show the experimental setup and the evaluation results, respectively. Chapter 5 explores new approach to improve the unfairness mitigation of code reviewer

recommendation systems. Chapter 6 discusses the threats to validity of this work. Finally, Chapter 7 concludes this work.

Chapter 2

Literature Review

2.1 Code Review

In software development, code review (CR) refers to inspecting code modifications manually for conformance to the team's quality standards. [35]. Typically, code inspection involves developers other than the original author reviewing the code, a practice widely acknowledged for its effectiveness in reducing software defects and enhancing overall product quality [36, 37, 38]. Moreover, researchers have demonstrated that code review activities not only impact software product quality but also have a significant effect on other aspects, such as software security [39, 40].

2.2 Modern Code Review

Modern code review (MCR) refers to the use of advanced tools, methodologies, and practices for reviewing code changes in software development. It evolved from traditional manual code review practices, which were often time-consuming. Modern code review is informal in structure, tool-based, and occurs on a regular basis in practice [36, 41, 39, 42, 43].

In modern code review (MCR), two primary groups of individuals are involved: 1) code authors and 2) code reviewers. They engage in activities that can be categorized into two main phases and six primary steps, which are described as follows [27, 44]:

- **Phase 1: Review planning and setup**
  - Step 1: The code authors prepare the code change for review, typically including a description of the intended modification and a reference to the corresponding issue.
  - Step 2: Assigning code reviewers based on different heuristics or tools to review the code change.
  - Step 3: Notification to code reviewers to inform them about the review and their duties.
- **Phase 2: Review execution**
  - Step 4: Code reviewers start to check the code changes for improvement or fixing the issue.

- Step 5: Code reviewers discuss the code change and the issue with the authors to get feedback on the problem.
- Step 6: The code change will be accepted/merged, rejected, or returned to the authors for refinement.

While various projects may differ in their code review practices and the extent to which they require it, there is generally little variation in how they approach code changes, with many following similar principles across different domains. [45]. In open-source development, reviewers prioritize building relationships with core developers, whereas, in commercial development, the dissemination of knowledge through MCR is more emphasized [27].

2.3 Code Reviewer Recommendation Systems

A code reviewer recommendation (CRR) system is a software application that supports software development teams in identifying the most appropriate code reviewers for a particular code change request by suggesting a list of the most qualified candidates to conduct the review request [41].

While the first code reviewer recommendation system already utilized machine learning techniques when it was introduced [46], earlier systems often relied on heuristic approaches [47, 48], such as graph and search-based approaches. As an example, Ounti et al. [47] proposed RevRec, a recommendation system that uses a genetic algorithm to find an appropriate peer reviewer for a code change. As the field progressed, newer systems increasingly employed more

machine learning methods [49, 50, 51, 6] e.g., SVM (Support Vector Machines), collaborative filtering, and Naive Bayes, to improve their recommendations. These reviewer recommendation systems use different factors and features for determining the most qualified reviewer for a review request, such as file similarity, developers' expertise, social relations, developers' activeness, etc [41].

In this work, we select two state-of-the-art ML-based code reviewer recommendation systems, i.e., RevFinder [29] and CORMS [6], as our research subjects to explore their fairness (details are in Section 3.2). These code reviewer recommendation systems have been shown to outperform their prior baselines and have been found effective in recommending code reviewers.

2.4 Fairness in Machine Learning

2.4.1 Definition of Fairness

In machine learning, fairness analysis is the process of figuring out if the algorithms, models, or systems that are being built are fair and treat everyone in a similar way, no matter their human-related or demographic information such as race, gender, or age [7, 8]. The aim is to ensure that the systems do not reinforce existing biases or discrimination and to prevent negative impacts on individuals or groups due to these biases [7, 8, 52, 53].

2.4.2 Sensitive/Protected Attribute

A sensitive or protected attribute divides the training examples into two groups: privileged (also known as unprotected) and unprivileged (also known as protected). This attribute usually comes from the human-related or demographic aspects of different groups of people, such as gender, race, and age. If a system treats people differently based on these attributes, the system is vulnerable to unfairness [7, 17, 8].

2.4.3 Types of Fairness

In general machine learning applications, there are two common types of fairness [8, 7]:

- Group Fairness: This type of fairness ensures that different groups of people, including protected groups and privileged groups, are treated similarly and fairly. For example, in cases where gender is not a deciding factor, the female group should be treated similarly to the other groups (e.g., the male group). [8, 7].

- Individual Fairness: This type of fairness ensures that individuals that are similar based on a criterion should be treated fairly and similarly [8, 7, 52]. For example, regardless of demographic background, each applicant in the employment process should be treated equally.

There is also another type of fairness known as "Subgroup Fairness", which attempts to combine the best characteristics of group and individual concepts of fairness [54, 55]. For the sake of brevity, we did not include this special case in the categorization. Also, it is worth

noting that it can be difficult to ensure all fairness definitions at the same time, especially when they may come into conflict with one another [7, 8].

In this study, we conduct our analysis based on the group fairness definition. Rather than focusing on individual cases, group fairness analysis evaluates the influence of machine learning models on distinct groups of people. This means that group fairness analysis can assist in identifying and mitigating biases that may not be visible at the individual level but have a major influence on specific groups of people [8, 56, 7].

2.4.4 Major Root Cause Categories

According to the literature [8, 7], the causes of unfairness in machine learning applications can be divided into two major categories:

- Data bias: unfairness can result from the presence of bias and prejudice in the dataset used to train machine learning algorithms. This can happen if the dataset contains skewed judgments, reports, and measures, an imbalanced attribute distribution, proxy attributes[1], or missing values.

- Algorithm bias: the algorithm itself has the potential to be unfair. The learning process may become biased and unfair if the model is optimized for performance at the expense of other factors.

[1]Proxy attributes are non-sensitive attributes used to train the model that are correlated with sensitive attributes.

2.4.5 Origins of Unfairness

Unfairness can exist in machine learning applications and can originate from various sources:

- Unfairness may exist in the datasets used to train the model. These datasets may contain the results of skewed judgments, reports, measures, and an imbalanced attribute distribution.
- Unfairness can also exist in the dataset due to missing values. In this case, the dataset will be unrepresentative due to the missing values.
- There is also the risk of unfairness caused by the algorithm. When the model is tuned for performance at the expense of other factors, the learning process can become biased and unfair.
- Unfairness can emerge implicitly from non-sensitive features/attributes in addition to sensitive features/attributes. The reason for this type of unfairness is that based on their values in the dataset, these attributes can become correlated with the sensitive ones. These are referred to as "Proxy Attributes."

2.5 Fairness in Recommendation Systems

In this work, we focus on the fairness of recommendation systems. The concepts and definitions of fairness analysis in recommendation systems and general ML applications slightly differ [56]. For example, in previous studies [7, 17, 8], since all approaches target

classification problems, the concept of fairness thoroughly depends on the predictions of the target attribute and its relation to the protected group. On the contrary, the concept of fairness in recommendation systems can be discussed from several points of view, e.g., the fairness of each of the recommended items (item-based fairness) and the fairness of the exposure and quality that each user experiences from the recommended items (user-based fairness) [56].

Also, in recommendation systems, not all of the biases are considered as unfairness, e.g., popularity bias, position bias, and conformity bias [56, 57]. the term "bias" has been used interchangeably with "unfairness" and "discrimination" to refer to systemic flaws or bias in algorithms or models that might result in unfair treatment of specific groups or individuals. However, not all biases are considered fairness problems in recommendation systems. [57, 56]. Nonetheless, it is noteworthy that in this study, terms such as "fairness issue or problem", "bias", "unfairness", and "unfairness issue or problem" are used interchangeably to illustrate the issue of fairness in code reviewer recommendation systems, as our study goal in this work is to examine the fairness problems.

2.5.1 Root Causes of Bias

Recommendation systems, like any other machine learning application, can suffer from bias [57, 7]. The following factors, in general, are the root causes of bias in recommendation systems [57]:

- The bias may come from user behavior data due to its observational nature instead of being more experimental.
- The bias may result from the uneven presentation of items in the dataset as a result of the varying popularity of multiple items and their user attention.
- The bias may be caused by recommendation system exposure mechanisms, which can determine user behaviors and amplify their choices back into the training process.

2.5.2 Types of Bias

Thus, according to [57], we can summarize the types of bias in recommendation systems into the following categories:

1. Bias in the data, which includes conformity bias, position bias, selection bias, and exposure bias.
2. Bias in the model, which includes inductive bias.
3. Bias in the results, which includes popularity bias and unfairness.

These various types of bias can also interact and exacerbate each other because of feedback loop mechanism in recommendation systems. However, in this work, we will mainly investigate the third category that includes unfairness.

2.5.3 Types of Fairness

The fairness of recommendation systems can be categorized into two groups [56]:

- Process fairness: This means ensuring that the process used to produce recommendations is fair and unbiased to all users, regardless of their sensitive attributes.

- Outcome fairness: This means ensuring that the system's outcomes are distributed fairly and proportionately among various groups of people. This means that recommendations should neither favor nor discriminate against any particular group based on their sensitive attributes.

The two groups might be seen from several perspectives concerning fairness issues [56]. For example, in recommendation systems, fairness can be assessed by looking at users, items, or both. These would correspond to review requests, reviewers, or both in code reviewer recommendation systems.

In this study, we focus on investigating **outcome fairness** in ML-powered code reviewer recommendation systems, specifically we examine the item-based fairness through **group fairness** analysis. In particular, the items being recommended in our case are code reviewers. Therefore, our research focuses on assessing whether the final list of recommendations upholds fairness with regards to these recommended code reviewers.

2.6 Unfairness Mitigation Techniques

To mitigate the unfairness in ML applications, many unfairness mitigation mechanisms have been proposed [8, 7], which can be categorized into three major types.

2.6.1 Pre-Processing Techniques

Pre-processing strategies [33, 32] aim to modify training data before it is utilized for training. The data itself is a common source of prejudice or unfairness. We may alleviate this unfairness even before the model attempts to learn it by applying pre-processing approaches.

2.6.2 In-Processing Techniques

In-processing approaches [58, 59] alter the machine-learning algorithm to increase fairness. For example, regularization can be used as one of the in-processing strategies to increase fairness in machine learning models [56, 7]. In-processing approaches can help decrease discrimination and ensure that the model delivers fair results for all groups by introducing fairness constraints into the model training process. Otherwise, the model will be penalized for unfairness.

2.6.3 Post-Processing Techniques

In a post-processing technique [60, 61], the output scores and predictions of the machine learning model are processed after the learning process has taken place in order to make them more fair.

In this work, we focus on the "post-processing" category of fairness mitigation methods and exclude both "in-processing" and "pre-processing" mitigation approaches. "in-processing" approaches require non-trivial changes to the code reviewer recommendation systems, which are not generalizable. Additionally, the majority of "pre-processing" approaches are unsuitable for

the recommendation tasks employed in our study subjects, given the inherent characteristics of our data, thus not suitable for fairness study in recommendation systems [33, 32].

2.7 Trade-Offs in Fairness Analysis

There are generally two main trade-offs in fairness analysis of ML-based applications:

2.7.1 Fairness Measures Trade-Offs

Past studies have revealed a challenge in achieving fairness across various measures and metrics [8, 7]. This difficulty arises from the fact that these measures are often incompatible, which is mainly due to the fact that fairness can be evaluated from different angles and defined in multiple ways. Moreover, the calculations involved in these measures can conflict with each other [7]. For example, enhancing fairness according to one measure might result in reduced fairness according to another measure. Therefore, choosing the right fairness measure that aligns with the context of our experiments and study is a critical step in conducting a fair analysis of machine learning-based applications [8, 7].

2.7.2 Fairness-Accuracy Trade-Off

Improving the fairness of ML-based applications is a challenging task. It requires a balance between enhancing fairness while maintaining accuracy. However, if we focus on making them fairer, it may lead to some accuracy being sacrificed [7, 8, 62]. Previous research by Corbett-Davies et al. [63] and Lipton et al. [64] has demonstrated this trade-off between

fairness and accuracy through theoretical analysis. Several empirical studies have also supported the existence of this trade-off [65, 66, 67]. Therefore, when evaluating unfairness mitigation techniques, we must also evaluate their impact on performance since they may have a trade-off.

2.8 Fairness in Software Engineering

Many studies have been done on addressing the problem of fairness in general ML-based applications and software engineering domain [25, 68, 69, 21, 17, 70]. Brun and Meliou [25] outlined a vision for how software engineering research may address fairness flaws and called on the software engineering research community to act. They argued that software fairness is identical with software quality and that different software engineering difficulties, such as requirements, specification, design, testing, and verification, must be addressed in order to tackle this problem. Albarghouthi et al. [68] simplified the problem to a series of weighted volume computation problems and solved them using an SMT solver. They further developed a tool, i.e., FairSquare, to validate the fairness qualities of decision-making programs derived from real-world datasets. Chakraborty et al. [21] introduced an unfairness mitigation approach that integrates both in-processing and pre-processing methods to eliminate ethical bias from the training data, leading to a fair trained model. Li et al. [17] proposed a pre-processing approach that finds biased portions of attributes correlated with the sensitive attribute and eliminates them from the training data, thereby enhancing fairness. Ferrara et al. [70] conducted an extensive survey into the perception and practical management of fairness in

software engineering by analyzing practitioners' viewpoints regarding the existing processes employed to handle fairness in software development, showing that fairness still remains a secondary consideration in developing artificial intelligence systems. Some other research studies [69, 71] work on the issue of bias in the process of modern code review. German et al. [69] investigated whether modern code reviews are conducted fairly, using fairness theory to develop a framework for understanding how fairness affects code reviews. Sultana et al. [71] demonstrated that female reviewers participate less in code reviews compared to males in a biased way. Additionally, many of them opt for gender-neutral profiles to increase the likelihood of their code being accepted. There also exist some other studies that involved bias and fairness in code reviewer recommender systems from different perspectives compared to our work [72, 73]. Tecimer et al. [72] worked on fixing bias caused by labeling errors to improve the performance of code reviewer recommendation. Murphy-Hill et al. [73] investigated systemic gender bias in code review loads at Google, examining gender-based discrimination similar to prior research in the field. They also demonstrated that integrating code review data into a recommendation system can amplify pre-existing human biases observed in the code review process, leading to biased decision-making when using these tools to help the manual reviewer selection process. That being said, in our study, we focus specifically on the fairness of ML-based automatic code reviewer recommendation systems, and unlike previous studies, we do not focus solely on the process of modern code review or the biases unrelated to fairness issues.

Chapter 3

Research Design

For our analysis, we evaluate whether the recommendations made for a specific group of reviewers (i.e., male or female) are fair and, if not, how we can mitigate the issue. Figure A.4 depicts the overview of our study.

This section describes our research questions (Section 3.1), study subjects (Section 3.2), experiment configurations (Section 3.3), data collection (Section 3.4), evaluation measures (Section 3.5), and the details of our selected unfairness mitigation strategies (Section 3.6).

3.1 Research Questions

In this study, we are going to answer the following three research questions (RQs):

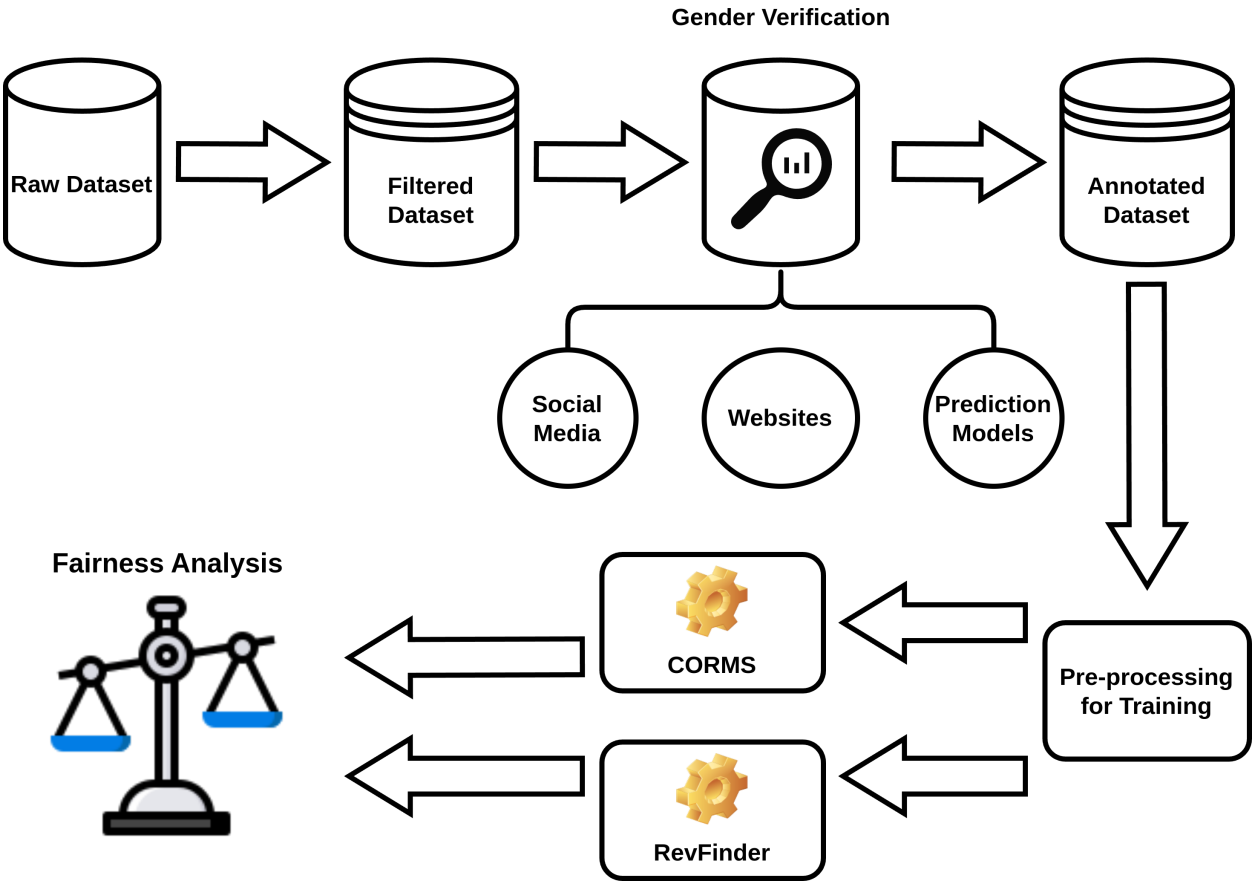


Figure 3.1: Overview of our conducted empirical study

3.1.1 RQ1: Existence of Unfairness

This RQ analyzes fairness in two typical ML-based code reviewer recommendation systems using benchmark data from four open-source software repositories (details are in Section 3.4). The findings of this section demonstrate the occurrence of discriminatory behavior and unfairness in code review systems in SE domain (details are in Section 4.1).

3.1.2 RQ2: Root Causes of Unfairness

In this RQ, we aim to investigate the factors that lead to unfairness in the results of recommendations in the studied subjects. These factors can come from several sources, such as an imbalanced dataset and popularity bias (details are in 4.2).

3.1.3 RQ3: Effectiveness of Existing Unfairness Mitigation Techniques

In this RQ, we examine the effectiveness of existing unfairness mitigation solutions in code review recommendation systems. Specifically, we apply current fairness-improving strategies in the literature to the two studied ML-based code reviewer recommendation systems and further check whether they can help mitigate the unfairness problems (details are in Section 4.3).

3.2 Subjects of Study

As mentioned in Section 2.3, we use two state-of-the-art ML-based code reviewer recommendation systems as the research subjects, i.e., CORMS [6] and RevFinder [29]. RevFinder recommends code reviewers based on a machine learning-based ranking algorithm that learns the scores of different candidates for a given review request based on the similarities in the file locations involved in past code reviews from the dataset. It employs four distinct file path comparison algorithms to compute similarities: 1) Longest Common Prefix, 2) Longest Common Prefix, 3) Longest Common Substring, and 4) Longest Common Subsequence.

RevFinder generates a list of unique reviewers and their associated scores by examining the dataset's records and computing the scores by propagating the scores of each reviewer involved in review requests with similar file path locations. Each file path comparison approach outlined earlier will be employed to generate a separate list of scores for the likely candidates for the code review recommendation. The outcomes of these four lists will then be combined using the Borda Count approach [74] to create a single list of scores for the recommendation process. By evaluating the records in the dataset and propagating the scores of each reviewer involved in review requests with similar file path locations, RevFinder provides a list of unique reviewers and their scores for the given dataset. On the other hand, CORMS uses the same similarity model as RevFinder (with additional features) and also employs a SVM model that learns from each review subject in the training set. Both systems apply combination techniques to combine the results into a single score list. Finally, appropriate reviewers are chosen based on their computed scores and associated ranks corresponding to their scores in the final recommendation list.

While both approaches have proven effective, fairness analysis has yet to be conducted on these systems. Although these two code reviewer recommendation approaches that we have examined can achieve good performance in terms of accuracy of the reviewer recommendation task, none of the fairness characteristics, e.g., race, age, and gender, were considered when recommending reviewers by using these approaches.

Moreover, there are two reasons that we selected these two code reviewer recommendation systems as the subjects of our study:

1. It is important to note that not all datasets suitable for training code reviewer recommendation systems are compatible with one another. This means that even if a particular dataset is useful for training one code reviewer recommendation system, it may not be appropriate for training a different code reviewer recommendation system.

2. Some datasets lack additional information or any human-related information regarding reviewers, making it difficult to conduct a fairness analysis. Therefore, we need to select subjects that are compatible with datasets that contain human-related information. Alternatively, we can annotate or augment human-related information into datasets lacking such information to facilitate our analysis.

3.3 Configurations for Fairness Analysis

This section describes the configurations we use to conduct our fairness analysis experiments.

3.3.1 Sensitive Attribute Selection

Our work focuses only on a demographic and human-related fairness factor, i.e., gender, as the attribute of interest for fairness analysis. This implies that our evaluation of fairness does not take into account attributes unrelated to humans, such as projects and file paths. The reason for this approach is that fairness issues in ML-based systems predominantly affect humans and their interactions, and we aim to address and understand these specific problems. Also, it is worth noting that our choice of "gender" as the sensitive attribute of our study is primarily

driven by the challenges associated with collecting other types of sensitive features (e.g., age, race, etc.) in our study domain. Additionally, according to the existing literature, gender is widely recognized as the most prominent and commonly addressed factor in addressing the diversity crisis [16]. Following previous fairness analysis studies [17, 7, 21, 20, 8], we limit our analysis to binary gender values of male and female because of the difficulty of collecting non-binary gender data, being in line with previous fairness research that used gender as their sensitive attribute, and for simplicity's sake due to the extendability of our fairness analysis to non-binary genders in the case of having sufficient data. In this study, the female reviewer group is considered the protected group.

3.3.2 Replication Configuration

Furthermore, we ensure the validity of our findings by replicating the configurations of the selected two code reviewer recommendation systems as specified in their respective publications [29, 6]. The results of replication are confirmed by comparing them to the results of the papers, and the authors of these works ensure an accurate presentation of their work. When training each model, following existing studies [29, 6], we divide the dataset into two sections, with 80% for training and the remaining 20% for testing, chronologically. In this work, we run CORMS and RevFinder on a PC equipped with a 2.8GHz i7-7700HQ processor and 16GB of RAM.

3.4 Data Collection

For our analysis, we need datasets that include human-related information of reviewers, such as their gender, which is our desired sensitive attribute. This information is essential for our study since it allows us to analyze statistics and measures related to protected groups. However, existing code reviewer recommendation datasets from previous studies [6, 29, 56] do not include that critical information. Thus, we must figure out how to augment the datasets with additional information, such as gender. To collect the gender information of reviewers, in this work, we propose heuristic methods to infer a reviewer's gender information from their names and other publicly available information online (e.g., homepage, GitHub account information, and LinkedIn profile). Our experiment is conducted on the same dataset as the previous study [6], which includes code review requests from **34 open-source projects**. These code review requests are collected from code review platforms like Gerrit and GitHub that are available to the public. As mentioned before, these code review systems do not keep reviewers' human-related information, such as gender, and we cannot directly obtain the genders of the reviewers from the datasets.

For each project in our dataset, we first get reviewers' names through the user ID provided by the code review platform. Then we use the following steps to infer a reviewer's demographic gender.

1. We first remove projects that have a high percentage of reviewers with missing names. If the field for a reviewer's name is null or blank, we consider this as not having a name mainly because the reviewer did not specify a name on his/her profile. Also,

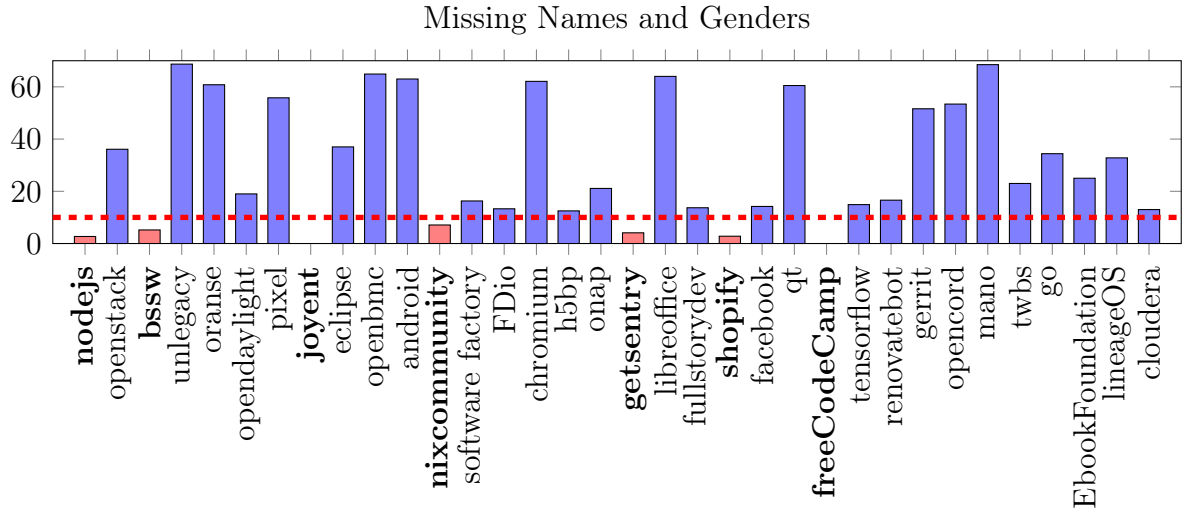


Figure 3.2: Missing rates of names and genders for each project in our study

reviewers may have nicknames instead of their real names on their profiles. We consider all these records "unknown" since we cannot infer the gender from them. To distinguish nicknames from real names, we use the following heuristic, i.e., if a name contains any number, symbol, or sign, we consider that a nickname; e.g., in the nixcommunity project, there was a reviewer with the name "jD91mZM2", which has been removed from the data. We discard projects with reviewers who have more than 10% "unknown" names to ensure the quality of our experiment data. The percentages of missing names and genders ("unknown") for each project examined in our study are demonstrated in Figure 3.2. We select projects with at most 10% missing names and genders, which is any project whose bar is beneath the red dashed line. As a result, we selected seven out of 34 projects. In the seven projects examined, there were at most four reviewers who had nicknames on their profiles.

2. Second, for the selected seven projects, we manually check each reviewer's information through online resources, e.g., personal and institutional homepages, GitHub accounts, and LinkedIn profiles. Some of the reviewers have links to their social media accounts on their GitHub profiles, so we can access their information directly. For the reviewers who do not provide social connections (20% of our reviewer set), we search their full name on the internet to find their social media accounts and get the gender information. During the process, we discard names that point to indistinguishable users (e.g., people who have the same name and similar profiles on social media). We are able to do this since we have the reviewers' names and can search for their profiles on the internet to gather information. We do this by searching for the reviewer's name plus the keywords related to the examined project on the internet. For example. If a reviewer's name is John Smith in the Node.js project, we use lookup queries like "John Smith Node.js", "John Smith Node.js LinkedIn", and "John Smith Node.js GitHub" on search engines. We look for gender information in their specified pronouns in their profiles, their online content, including the pronouns and genders the reviewers themselves or others used to describe them. In this manual analysis, we checked 355 reviewers' information on the internet. These reviewers may possess several reviews and contributions in our dataset, employed for training the models, and the corresponding statistics are displayed in Table 3.1. According to our analysis, we obtained at least 90% of the reviewers' gender information through this manual analysis. Three authors are involved during this process to ensure the correctness of the manually analyzed results.

Table 3.1: Details of the selected projects

Project	Missing Genders	Total Ex.	F.Ratio	F#	M#
nodejs	2.7%	110	0.13	5	28
bssw	5.2%	213	0.67	9	9
getsentry	4.1%	775	0.08	6	64
shopify	2.8%	565	0.06	17	148

3. Finally, we remove projects for which there is only one reviewer from the protected group, i.e., the female reviewer group. In each remaining project, we exclude records relating to reviewers of unknown genders.

After applying our gender identification approach, four out of the 34 candidate projects from [6] are selected for our experiments, as demonstrated in Table 3.1. In this table, the columns "Missing Genders", "Total Ex.", "F. Ratio", "M#", and "F#", are the percentage of missing genders, the total number of examples (total number of reviews), the ratio of female reviewers, the number of female reviewers, and the number of male reviewers, respectively. Our strategy ensures the reliability and validity of our results by guaranteeing that each of the projects we use contains at least 90% of its total reviewers and multiple people from different gender groups.

3.5 Evaluation Measures

This section describes our evaluation measures for fairness and recommendation performance. Existing studies [64, 10] revealed that improving fairness usually comes at a cost in terms of performance measures such as model accuracy. As a result, in order to demonstrate that an

unfairness mitigation strategy is useful to deploy, most studies in fairness analysis will include a performance measure before and after applying the unfairness mitigation technique [7, 17, 63]. It is important to clarify that the objective of our study is to evaluate the fairness of code reviewer recommendation systems, rather than focusing on their performance. Performance measures are solely utilized to assess the influence of unfairness mitigation strategies on the recommendation outcomes, both before and after implementing these strategies. In this work, we follow existing studies [17, 7] and use the average of all measure values for each record in our dataset to represent the overall performance.

3.5.1 Fairness Measures

To evaluate the fairness of recommendation systems, we use two measures that rely on the top-K results and one measure that is independent of the top-K results [31, 56]. Both measures are useful for analyzing group fairness, which is the goal of our investigation. Similar to existing works [6, 29] that conducted their experiments in a specific setting, we limit the values of K to 4, 6, and 10.

- $Skew_{\mathrm{S_i}}@K$: The skew of the ranked list of top-K candidates for a certain value S_i of the sensitive attribute is:

$$Skew_{\mathrm{S_i}}@K(C) = \ln(\frac{P_{\mathrm{C^K,S_i}}}{P_{\mathrm{D,S_i}}}) \tag{3.1}$$

where C is the ranked list of candidates, C^K is the top-K candidates from C, $P_{\mathrm{C^K,S_i}}$ is the proportion of candidates having the sensitive attribute value S_i in the top-K results, and $P_{\mathrm{D,S_i}}$ is the desired proportion of candidates with the sensitive attribute value S_i in the given dataset. The desired proportion for each project is calculated by dividing the number of female reviewers by the total number of reviewers. When the proportion of the protected group is smaller than the desired proportion, the fraction's output is less than one, causing the natural logarithm's output to be negative. As a consequence, if this measure returns a negative value, we have a fairness problem for the sensitive attribute value S_i and the top-K recommendation list is not a fair recommendation. The ideal value of this measure is zero (since the output of the fraction will be one and the natural logarithm of one is zero). The closer the value of this measure comes to zero, the more fair the recommendation. Yet, if the measure produces a positive result, it indicates that the recommendations favor the protected group [31].

- Statistical Parity Difference For top-K results ($SPD@K$): The statistical parity difference (SPD) is a well-known measure of fairness that is used in many articles about how fair machine learning is when it comes to classification tasks [7, 17]. This measure is demonstrated in Eq. 3.2 as an example for the binary classification:

$$\left| P[\hat{Y} = 1 | S = 1] - P[\hat{Y} = 1 | S \neq 1] \right| \leq \epsilon \tag{3.2}$$

This measure implies that the rate or probability of a positive prediction for the protected group should be similar to that of the privileged group. In the best-case scenario, these probabilities or rates are equal to one another, and the SPD value is zero. Yet in reality, to consider the result of a classification algorithm fair, the difference between these probabilities should be less than a threshold known as epsilon (ϵ), which should be chosen based on the problem information.

Nevertheless, in order to use this measure for top-K results in recommendation systems such as code reviewer recommendation systems, we must change the calculation in Eq. 3.2. As a result, we introduce the $SPD@K$ measure, which is described in Eq. 3.3:

$$\left| P[\hat{Y} \in C^K | S = 1] - P[\hat{Y} \in C^K | S \neq 1] \right| \leq \epsilon \tag{3.3}$$

Where C^K is the ranked list of top-K candidates from the ranked list of candidates C as the result of the recommendation. In this measure, we also refer to ϵ as the expected SPD threshold. This threshold will be computed by calculating the absolute difference between the male and female ratios in the dataset, which is described in Eq. 3.4:

$$\epsilon = \left| \frac{\#\text{ Females}}{\#\text{ Reviewers}} - \frac{\#\text{ Males}}{\#\text{ Reviewers}} \right| \tag{3.4}$$

We expect that the difference in recommendation rates for males and females will be similar to the disparity in ratios of these groups in the dataset.

- Normalized Discounted Cumulative KL-divergence (NDKL): Eq. 3.5 describes the normalized discounted cumulative Kullback-Leiber (KL) divergence given a ranked list of the candidates C:

$$NDKL(C) = \frac{1}{Z} \sum_{i \in Ks} \frac{1}{\log_2(i+1)} d_{\mathrm{KL}}(D_{\mathrm{C^i}} || D_d) \tag{3.5}$$

In Eq. 3.5 $d_{\mathrm{KL}}(D_{\mathrm{C^i}} || D_d) = \sum_j D_{\mathrm{C^i}}(j) \ln \frac{D_{\mathrm{C^i}}(j)}{D_d}$, $Z = \sum_{\mathrm{i=1}}^{\mathrm{C}} \frac{1}{\log_2(i+1)}$, and $Ks = \{4, 6, 10\}$ where $D_{\mathrm{C^i}}$ and D_d represent the proportion of the top i candidates in the ranked list of candidates C having the sensitive attribute j, respectively, and the desired proportion based on the dataset with the sensitive attribute value j.

This measure considers all sensitive attribute values, which means it considers both privileged and underprivileged groups [31, 56]. It is also independent of the value of K, unlike the other measures, because its value accumulates over different K values. This measure has non-negative values, with zero being the optimal value for a fair recommendation based on it. As a result, if the value is positive, it indicates that the recommendation has fairness issues. The problem with this measure is that we cannot be sure which attribute value is causing the recommendation to be unfair, and it's difficult to interpret. The most effective way to use it is to look for a decrease in the values of this measure, which indicates that fairness has improved [31].

3.5.2 Performance Measures

In this study, we adopt the identical performance metrics utilized in previous research papers discussing CORMS and RevFinder [6, 29], i.e., Top-K Accuracy and Mean Reciprocal Rank (MRR). This enables us to cross-compare and guarantee accurate replications of both works and to ensure a fair assessment before and after applying bias mitigation strategies. Top-K accuracy is a performance measure that shows how often a code reviewer recommendation system can suggest the right code reviewers for a set of reviews [6, 29]. It is calculated by dividing the number of reviews where at least one correct code reviewer was in the top-K recommendations by the total number of reviews. MRR is defined as the average of the reciprocal ranks of the correct answers in the ranked list of recommended code reviewers [6, 29]. In other words, for each review, MRR calculates the reciprocal rank of the first relevant result and averages these values over all the reviews in the dataset. In our scenario, we use $MRR@K$ to focus on the top-K recommendations. This means that we only consider the top-K candidates in the original calculations. Every candidate not on the top-K list has a reciprocal rank of zero. The rest of the calculations remain the same.

3.6 Selected Unfairness Mitigation Techniques

Researchers have proposed several unfairness mitigation strategies for recommendation systems [56, 30, 75, 31, 76, 77, 78], but not all of them are applicable to our case due to different views on fairness [56]. Also, due to the inherent differences in the fairness of

classification and recommendation tasks (e.g., techniques used for classification do not support top-K results), we cannot employ those mitigation techniques for classification tasks in our study. Moreover, each recommendation system can have a distinct and different architecture, and only some mitigation techniques apply to that type of architecture (e.g., reinforcement learning-based approaches may not be suitable for many recommendation systems) [56].

In this work, we select two applicable approaches from the "post-processing" category of fairness mitigation methods (see 2.6) for reviewer recommendation systems [56]. We did not apply any "in-processing" mitigation approaches as these approaches require non-trivial changes to the code reviewer recommendation systems, which do not generalize. In addition, we exclude "pre-processing" mitigation techniques, as these data-oriented techniques in the literature are primarily designed for classification tasks and, thus, are unsuitable for our study. Also, the training data used by our study subjects is not entirely compatible with the pre-processing techniques available in the literature for fairness of recommendation systems, leading to their opting out of our analysis [33, 32]. The details of these two mitigation approaches are as follows.

3.6.1 DetGreedy

Geyik et al. [31] developed this algorithm for fairness-aware recommendation in LinkedIn Talent Search recommendation systems. This algorithm works as follows. Given a top-K list, there are two requirements to satisfy the fairness condition:

a. Min: $\forall K < |C| \,\&\forall s_i \in A, count_K(S_i) \geq \lfloor P_{D,S_i} \cdot K \rfloor$

b. Max: $\forall K < |C| \,\&\forall s_i \in A, count_K(S_i) \leq \lceil P_{D,S_i} \cdot K \rceil$

Where A is the set of attribute values, C is the list of candidates, P_{D,S_i} is the desired proportion of candidates with the attribute value S_i, and $count_k(S_i)$ is the number of candidates with the attribute value S_i in the top-K results. If there are candidates whose attribute values are close to violating the minimum requirement, choose the candidate with the highest score next from that group. If all candidates meet the minimum requirement, then pick the one with the highest next score from the group based on the attribute value among those who have not yet reached their maximum requirements.

3.6.2 DetRelaxed

To improve DetGreedy, Geyik et al. [31] further proposed DetRelaxed. While DetGreedy aims to include as many high-scoring candidates as possible in the ranked list, it may not be effective in various scenarios, according to the authors [31]. For example, DetGreedy can easily fall into an infeasible state, which is a state that breaks the fairness condition in the minimum representation explained above, according to the authors. The DetRelaxed algorithm was proposed to consider all candidates who satisfy the minimum requirement and minimize the term $\left\lceil \frac{\lceil P_{D,S_i} \cdot K \rceil}{P_{D,S_i}} \right\rceil$ to select the candidate with the highest score for the next position.

According to the previous study [31], these approaches resulted in a huge improvement in fairness at the production stage. Hence, we select these approaches to assess their effectiveness with our subjects.

Chapter 4

Results and Analysis

This section presents the experimental results and answers the research questions we asked in Section 3.1.

4.1 RQ1: Existence of Unfairness

Approach: To answer this RQ, we first build RevFinder and CORMS on the training data selected from our experimental dataset (details are in Section 3.4). When training each model, following existing studies [29, 6], we divide the dataset into two sections, with 80% for training and the remaining 20% for testing, chronologically. Then, we use the measures mentioned in Section 3.5.1 to evaluate the fairness of the recommendations generated by these code reviewer recommendation systems based on the testing data.

Result: Table 4.1 presents the findings of our experiments, where we analyze all the evaluation measures for each project across three different scenarios, i.e., original (i.e., results obtained

without any mitigation approaches), DG (i.e., outcomes obtained after using DetGreedy), and DR (i.e., results obtained after using DetRelaxed). Also, it presents the SPD threshold for each project in the dataset. As we can see from column "Original" in the table, $Skew@K$ has negative values under both CORMS and RevFinder on the four projects with different K values, e.g., BSSW, GetSentry, Node.js, and Shopify, which have negative $Skew@K$ values, i.e, -0.19, -0.01, -0.73, and -0.90 under CORMS when recommending top 10 reviewers. For $SPD@K$, we show the detailed desired SPD threshold of each project in the last row in Table 4.1. In this table, the results that are considered unfair recommendations based on the $SPD@K$ and $Skew@K$ measures are highlighted in • and •, respectively. As we can see on most projects, the values of $SPD@K$ are above the specified threshold values. For the projects BSSW, GetSentry, Node.js, and Shopify, the average variation between the $SPD@K$ measure values and the SPD threshold is 0.15, 0.03, 0.03, and 0.08, respectively. This indicates that the disparity between the percentages of males and females being recommended in BSSW, GetSentry, Node.js, and Shopify projects is 15%, 3%, 3%, and 8%, respectively. As an illustration, in the Shopify project's Top-6 scenario, the $SPD@K$ measures for both CORMS and RevFinder are 0.80 and 0.97 and are highlighted in yellow, respectively, which exceeded the anticipated SPD threshold of 0.79, indicating that the results were unfair. Overall, considering the different $SPD@K$ metric variations and their corresponding expected SPD threshold, male reviewers get an average of 7.25% more recommendations than female code reviewers compared to their distribution in the reviewer set. This calculation involves averaging across all the mentioned variations. Results from $Skew@K$ and $SPD@K$ indicate

there are unfairness issues for each of the four experimental projects under both CORMS and RevFinder.

Furthermore, we analyze the normalized discounted KL divergence measure (NDKL), which is shown in Table 4.1. Our experiments indicate that the NDKL measure yielded positive values for all projects, suggesting that the outcomes may be biased towards a particular value of the sensitive attribute. As discussed in Section 3.5.1, a disadvantage of this measure is its challenging interpretation due to the unknown sensitive attribute value that causes unfairness. However, we acknowledge that the recommendation could be unfair, which helps us address this research question.

Answer to RQ1: In the examined projects, male reviewers get an average of 7.25% more recommendations than female code reviewers, considering their distribution in the recommendation list under both CORMS and RevFinder. Our experiment results suggest that the two ML-based code reviewer recommendation systems studied exhibit bias towards gender.

4.2 RQ2: Root Causes of Unfairness

Approach: To answer this question, we first conduct an exclusive analysis of current fairness studies in the literature to collect all the possible factors that have been examined to be effective in determining the fairness of ML applications [8, 57, 7, 79, 56], especially recommendation systems. three factors were collected: (1) imbalanced or skewed data, (2)

Table 4.1: Summary of fairness analysis experiment results for RQ1

Top-K	Subjects	Measures	Projects											
			BSSW			GetSentry			Node.js			Shopify		
			Original	DG	DR	Original	DG	DR	Original	DG	DR	Original	DG	DR
Top-4	CORMS	TopK-ACC	79%	79%	79%	43%	43%	37%	41%	41%	41%	25%	25%	30%
		MRR@K	0.54	0.54	0.54	0.23	0.23	0.21	0.26	0.26	0.27	0.19	0.19	0.21
		SPD@K	0	0	0	0.93	0.93	1	0.85	0.85	1	0.80	0.87	1
		Skew@K	0.01	0.01	0.01	-1.82	-1.82	-2.25	-1.79	-1.79	-2.7	-1.41	-1.50	-2.35
	RevFinder	TopK-ACC	71%	73%	73%	43%	43%	37%	50%	50%	45%	25%	25%	24%
		MRR@K	0.5	0.5	0.5	0.25	0.25	0.22	0.32	0.32	0.31	0.15	0.15	0.14
		SPD@K	0.09	**0**	**0**	0.90	**0.88**	1	0.52	0.56	1	0.97	**0.96**	1
		Skew@K	-0.001	**0.01**	**0.01**	-1.48	-1.48	-2.12	0.15	**0.09**	-2.71	-2.03	**-1.94**	-2.17
Top-6	CORMS	TopK-ACC	84%	87%	87%	44%	44%	38%	52%	52%	47%	34%	36%	37%
		MRR@K	0.55	0.56	0.56	0.24	0.24	0.21	0.29	0.29	0.28	0.21	0.21	0.22
		SPD@K	0.31	**0**	**0**	0.87	0.87	1	0.86	0.86	**0.66**	0.80	0.87	1
		Skew@K	-0.35	**0.01**	**0.01**	-1.20	-1.20	-2.25	-1.56	-1.56	**0.12**	-1.14	-1.28	-2.35
	RevFinder	TopK-ACC	88%	83%	83%	49%	49%	46%	54%	54%	54%	36%	36%	37%
		MRR@K	0.53	0.52	0.52	0.26	0.26	0.24	0.33	0.33	0.33	0.17	0.17	0.17
		SPD@K	0.13	**0**	**0**	0.81	**0.80**	1	0.60	0.69	0.66	0.97	**0.96**	1
		Skew@K	-0.14	**0.01**	**0.01**	-0.53	-0.58	-2.12	0.07	-0.1	0.15	-1.97	**-1.87**	-2.17
Top-10	CORMS	TopK-ACC	94%	100%	100%	45%	46%	45%	58%	58%	58%	42%	43%	46%
		MRR@K	0.57	0.57	0.57	0.24	0.24	0.24	0.29	0.29	0.29	0.22	0.22	0.22
		SPD@K	0.2	**0**	**0**	0.79	0.82	0.8	0.82	**0.78**	**0.8**	0.77	**0.73**	0.8
		Skew@K	-0.19	**0.01**	**0.01**	-0.01	-0.1	**0.14**	-0.73	**-0.31**	**-0.35**	-0.90	**0.26**	**0.04**
	RevFinder	TopK-ACC	88%	92%	92%	62%	62%	55%	68%	68%	68%	44%	44%	43%
		MRR@K	0.53	0.53	0.53	0.28	0.28	0.25	0.35	0.35	0.35	0.18	0.18	0.17
		SPD@K	0.17	**0**	**0**	0.81	0.82	1	0.68	0.68	0.8	0.96	0.96	1
		Skew@K	-0.19	**0.01**	**0.01**	-0.03	-0.16	-2.12	0.06	0.06	-0.32	-1.83	**-1.77**	-2.17
	CORMS	NDKL	0.04	**0.01**	**0.01**	0.08	0.08	0.08	0.11	**0.10**	**0.08**	0.14	**0.08**	**0.09**
	RevFinder	NDKL	0.04	**0.01**	**0.01**	0.07	0.07	0.09	0.06	0.06	0.08	0.09	0.09	0.10
	SPD Threshold		0			0.82			0.69			0.79		

popularity bias, and (3) algorithmic objectives. We are not going to examine algorithmic objectives since it requires us to examine the code reviewer recommendation architectures, which is out of the scope of this work. As a result, we employ the factors of imbalanced or skewed data and popularity bias to explore the possible root causes of code reviewer recommendation systems.

Imbalanced or skewed data can lead to unfairness because models trained on such data have a strong probability of learning behavior towards over-represented groups, eventually becoming overfitted to them [56, 57]. Popularity bias and unfairness also have a strong connection with each other [57, 79]. The "long-tail effect" occurs when recommendation

systems favor popular items over less popular ones [80, 81], which leads to discrimination against the less popular items. If the less popular items are generally from the protected group, popularity bias will turn into unfairness.

Result: The following are the primary factors responsible for the unfairness and disparities between actual and expected distributions of the protected group in the outputs of the code reviewer recommendation system:

Imbalanced or Skewed Data: Our analysis indicates that the imbalanced representation of male and female reviewers in some projects is one of the reasons behind the unfair outcomes of code reviewer recommendation systems. Table 3.1 shows that, apart from the BSSW project, which has an equal number of female and male reviewers, the number of male reviewers is higher than that of female reviewers in all other projects (males are approximately 8 times more than females), which leads to skewed data. The BSSW project has a record of females accounting for 67% of the total examples, while in the Node.js, GetSentry, and Shopify projects, this percentage is 13%, 8%, and 6%, respectively. Consequently, Table 4.1 reveals that the results of the $SPD@K$ measure for the BSSW project are substantially lower than those of other projects, suggesting a fairer outcome. The same pattern is observed for the $Skew@K$ measure in the BSSW project, as other projects exhibit more negative values for this measure, indicating larger discrepancies between the current and fair conditions.

Popularity Bias: In the projects examined, bias was found to be more prevalent in projects where male reviewers were more popular than female reviewers. For instance, in Shopify and Node.js, the first 14% and 15% of popular reviewers were all male, respectively.

This led to unfairness in the CORMS and RevFinder recommendation systems, which learned and incorporated this preference for male reviewers into their decision-making processes. In contrast, in the BSSW project, the first two popular reviewers were female, resulting in fairness measures like $Skew@K$ and $SPD@K$ being closer to fair values, such as $Skew@K$ being closer to zero. More details can be found in the supplement materials (more details in Appendix A).

Answer to RQ2: We confirm that popularity bias and imbalanced or skewed data are two factors that can affect code reviewer recommendation systems, both of which can lead to unfairness. Projects that do not have these issues (e.g., BSSW) result in values in fairness measures that are closer to a fair state, in contrast to projects that do suffer from these problems.

4.3 RQ3: Effectiveness of Existing Unfairness Mitigation Techniques

Approach: In this RQ, we examine the selected unfairness mitigation approaches (i.e., DetGreedy and DetRelaxed) in Section 3.6 to see if they could improve the fairness of code reviewer recommendation systems. Furthermore, because applying an unfairness mitigation technique has been shown to have a trade-off with performance measures [63, 64], we should also guarantee that applying these mechanisms does not adversely affect performance measures.

As a result, we compare performance measures before and after using unfairness mitigation techniques.

Result: The fairness measures after applying these two mitigation approaches, i.e., DetGreedy and DetRelaxed, for each project, are shown in Table 4.1.

As we can see from the table, both DetGreedy and DetRelaxed mitigation techniques can improve fairness, but not across all projects. The bolded values on this table indicate fairness improvements compared to original settings. With the use of these approaches, the BSSW project saw a significant fairness improvement of 100%. Also, in the BSSW project, RevFinder's top-K accuracy decreased by 5% solely in the top-6 scenario. While performance measures were not adversely impacted (only a 1.75% decrease on average on all projects), the fairness enhancement was not as noticeable in projects with an imbalanced distribution of male and female reviewers (e.g., Node.js, GetSentry, and Shopify).

Answer to RQ3: DetGreedy and DetRelaxed mitigation approaches can improve fairness while maintaining performance, but not consistently across all projects.

Chapter 5

Addressing the Limitations of Current Mitigation Methods

5.1 Problem with Existing Mitigation Methods

As presented in Section 4.3, although both examined mitigation approaches, DetGreedy and DetRelaxed, can help improve fairness for some projects, we can also see that both DetGreedy and DetRelaxed [31] do not work effectively for certain projects. For example, the fairness improvement in the BSSW project is significantly higher than that of other projects e.g., Node.js, GetSentry, and Shopify. Our analysis shows that these two approaches can be effective when the data is not imbalanced or skewed in terms of protected and privileged group ratios. However, for projects with imbalanced or skewed data, these approaches may not be effective in improving fairness. As highlighted in Section 4.2, imbalanced or skewed

data is one of the main factors causing unfairness in such systems. Therefore, a solution that can address this issue is necessary for improving fairness.

5.2 Improved Re-Ranking Algorithm (IRR)

In order to tackle this problem, we propose an improved re-ranking algorithm. The motivation behind this algorithm is that it should guarantee to at least bring a reviewer from the discriminated group into the top-K list, regardless of other constraints. Although the DetGreedy and DetRelaxed algorithms specify the minimum and maximum requirements in Section 3.6, following both requirements as a fairness condition does not necessarily result in this outcome. Algorithm 1 demonstrates the algorithm of our approach.

The method aims to improve fairness in a recommendation list by using the $SPD@K$ measure. It replaces a female reviewer with the highest score not on the top-K list with a male reviewer with the lowest score on the top-K list to achieve the SPD threshold. This process is repeated until the threshold is met.

5.3 IRR's Unfairness Mitigation Evaluation

Table 5.1 shows the results of the experiments with our approach. The bolded values on this table indicate fairness improvements compared to the original settings. In this table, • indicates the results where our approach works but the DetGreedy and DetRelaxed methods did not succeed in achieving fairness improvements. • indicates that where our approach

Table 5.1: Results of unfairness mitigation effectiveness of IRR

Top-K	Subjects	Measures	BSSW	GetSentry	Node.js	Shopify
Top-4	CORMS	Top-K-ACC	53%	40%	41%	24%
		MRR@K	0.43	0.23	0.26	0.19
		SPD@K	0.5	**0.5**	**0.5**	**0.43**
		Skew@K	-0.65	**1**	**0.5**	**1**
	RevFinder	Top-K-ACC	57%	39%	50%	21%
		MRR@K	0.42	0.24	0.32	0.13
		SPD@K	0.5	**0.5**	**0.45**	**0.5**
		Skew@K	-0.55	**1.13**	**0.6**	**1.07**
Top-6	CORMS	Top-K-ACC	87%	43%	47%	32%
		MRR@K	0.56	0.24	0.28	0.2
		SPD@K	0.33	**0.66**	**0.66**	**0.59**
		Skew@K	**0.26**	**0.61**	**0.12**	**0.66**
	RevFinder	Top-K-ACC	76%	49%	54%	33%
		MRR@K	0.51	0.26	0.33	0.16
		SPD@K	0.33	**0.66**	**0.57**	**0.66**
		Skew@K	-0.1	**0.75**	**0.33**	**0.69**
Top-10	CORMS	Top-K-ACC	97%	45%	58%	40%
		MRR@K	0.57	0.24	0.29	0.22
		SPD@K	0.2	**0.77**	**0.6**	**0.53**
		Skew@K	**0.19**	**0.23**	**0.29**	**0.81**
	RevFinder	Top-K-ACC	88%	62%	59%	44%
		MRR@K	0.53	0.28	0.34	0.18
		SPD@K	0.2	**0.78**	**0.6**	**0.79**
		Skew@K	**0**	**0.31**	**0.32**	**0.22**
	CORMS	NDKL	0.09	**0.06**	**0.03**	**0.11**
	RevFinder	NDKL	0.09	**0.08**	**0.05**	**0.07**
	SPD Threshold		0	0.82	0.69	0.79

outperformed the DetGreedy and DetRelaxed approaches in terms of fairness improvements. The unfair recommendations based on the $SPD@K$ and $Skew@K$ measures are highlighted in • and •, respectively.

Despite the inability of DetGreedy and DetRelaxed to enhance fairness in the following scenarios, our approach significantly increased fairness by a greater margin (approximately

Algorithm 1 Improved Re-Ranking Algorithm (IRR)

```
1: procedure IRR(scores_list, SPD_threshold, K)
2:     topK_list = topK(scores_list, K)
3:     topK_excluded = exclude_topK(scores_list)
4:     other_females = get_females(topK_excluded)
5:     other_males = get_males(topK_excluded)
6:     while is_unfair(SPD(topK_list), SPD_threshold, K) do
7:         before_scores_list = topK_list
8:         m_val_before = metric(topK_list)
9:         if unfair_towards_males(topK_list) then
10:            substitute(topK_list, other_females)
11:        else
12:            substitute(topK_list, other_males)
13:        end if
14:        m_val_after = metric(topK_list)
15:        if m_val_after ≥ m_val_before then
16:            return sort(before_scores_list)
17:        end if
18:    end while
19:    return sort(topK_list)
20: end procedure
```

24% on average in all scenarios): four more scenarios in the GetSentry project (CORMS-Top-4, CORMS-Top-6, CORMS-Top-10, RevFinder-Top-10), three more scenarios in the Node.js project (CORMS-Top-4, RevFinder-Top-6, RevFinder-Top-10), and two more scenarios in the Shopify project (CORMS-Top-4, CORMS-Top-6). Additionally, while enhancing fairness, the average top-K accuracy was reduced by 1.3%, 2.6%, and 2% in the GetSentry, Node.js, and Shopify projects, respectively, across all scenarios.

To verify the significance of our approach's fairness improvement compared to the two baselines, i.e., DetGreedy and DetRelaxed, we employ the Wilcoxon signed-rank test [82] to test our null hypothesis. The null hypothesis being tested is whether there is a significant difference between the outcomes of our approach's improvement and those of the other two

methods. If the p-value is less than 0.05, it means that the two approaches that were compared have significantly different results. The outcomes of our tests on three projects (GetSentry, Node.js, and Shopify), where fairness was enhanced, demonstrate that the p-values for our approach compared to DetGreedy and DetRelaxed are 0.000007 and 0.0002, respectively. The results imply that our approach's improvements are statistically significant and not due to chance. In terms of fairness evaluation, our approach performs better than both the DetGreedy and DetRelaxed approaches in the examined scenarios.

Note that although our approach results in an overall better performance than both DetGreedy and DetRelaxed, it still cannot fully mitigate the recommendation unfairness on the BSSW project. For example, in this project under RevFinder's top-4 scenario, our approach's $SPD@K$ value is 0.5 (after mitigation), which exceeds the anticipated SPD threshold of 0, which indicates further efforts on improving the fairness for projects with different characteristics are needed.

Our proposed mitigation approach can significantly outperform the existing DetGreedy and DetRelaxed mitigation approaches.

Chapter 6

Threats to Validity

The main threat to the validity of this work is the generalizability, as both RevFinder [29] and CORMS [6] use a similarity-based model as a core part of their recommendation systems. They attempt to recommend reviewers based on the similarity of the files involved in previous reviews for a given new review request. Additionally, CORMS employs a hybrid approach that combines the results of the similarity model with an SVM model trained on the subjects of previous change reviews. Therefore, it is still possible that the results of experiments conducted on other code reviewer recommendation systems, such as those that use developer experience, could differ from those of CORMS and RevFinder. Consequently, our findings may not be generalizable to those systems. Future research should address these concerns with different code reviewer recommendation systems. Additionally, as stated earlier, to align with prior fairness studies that employed gender as the sensitive attribute [17, 7, 21, 20, 8], and considering the challenge of acquiring non-binary data, we conducted our analysis using

the binary gender attribute. However, it's worth noting that our analytical approach can accommodate attributes with multiple values, not solely limited to binary ones. Therefore, if adequate data on non-binary genders becomes available, our research can be extended to encompass those scenarios.

Chapter 7

Conclusion and Future Work

This study represents a novel investigation into the fairness issue of machine learning-based code reviewer recommendation systems. Specifically, two state-of-the-art systems (CORMS and RevFinder) and code review data from four open source projects were used to conduct the fairness analysis. Our empirical study demonstrates that current state-of-the-art ML-based code reviewer recommendation techniques exhibit unfairness and discriminating behaviors. This study also discusses the reasons why the studied ML-based code reviewer recommendation systems are unfair and provides solutions to mitigate the unfairness.

Moreover, based on existing literature[16], the diversity crisis emerges as a crucial concern within software development, constituting a multi-dimensional problem. Among these dimensions, gender stands out as the most prominent and applied one, particularly when addressing the need to increase diversity with a focus on women. Thus, our research holds promise as a valuable contribution to alleviating the diversity crisis in software development.

In the future, we plan to broaden our research scope by investigating more sensitive attributes, analyzing a wider range of code reviewer recommendation systems with different architectures, multi-objective search to make a recommendation that maximizes performance and minimizes bias, and examining fairness issues in other tasks in SE domain. To make that research possible, the real challenge will be to find relevant datasets. This calls for the creation of additional and more diverse datasets with several sensitive attributes, especially for ML-powered software engineering tasks since most of the datasets lack sensitive attributes on their data. We also plan on using automated methods to extract human-related data from code review repositories, which will support further research in the field of fairness analysis.

Bibliography

[1] Y. Yang, X. Xia, D. Lo, and J. Grundy, "A survey on deep learning for software engineering," *ACM Comput. Surv.*, vol. 54, no. 10s, sep 2022. [Online]. Available: https://doi.org/10.1145/3505243

[2] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 297–308. [Online]. Available: https://doi.org/10.1145/2884781.2884804

[3] S. Wang, T. Liu, J. Nam, and L. Tan, "Deep semantic feature learning for software defect prediction," *IEEE Transactions on Software Engineering*, vol. 46, no. 12, pp. 1267–1293, 2020.

[4] Z. Li and H. Zhong, "Revisiting textual feature of bug-triage approach," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2021, pp. 1183–1185.

[5] M. Chouchen, A. Ouni, M. W. Mkaouer, R. G. Kula, and K. Inoue, "Whoreview: A multi-objective search-based approach for code reviewers recommendation in modern code review," *Applied Soft Computing*, vol. 100, p. 106908, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494620308462

[6] P. Pandya and S. Tiwari, "Corms: a github and gerrit based hybrid code reviewer recommendation approach for modern code review." Association for Computing Machinery (ACM), 11 2022, pp. 546–557.

[7] D. Pessach and E. Shmueli, "A review on fairness in machine learning," *ACM Computing Surveys*, vol. 55, pp. 1–44, 4 2023.

[8] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan, "A survey on bias and fairness in machine learning," *ACM Computing Surveys (CSUR)*, vol. 54, 7 2021. [Online]. Available: https://dl.acm.org/doi/10.1145/3457607

[9] J. Chakraborty, K. Peng, and T. Menzies, "Making fair ml software using trustworthy explanation," *Proceedings - 2020 35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020*, pp. 1229–1233, 9 2020. [Online]. Available: https://doi.org/10.1145/3324884.3418932

[10] S. A. Friedler, C. Scheidegger, S. Venkatasubramanian, S. Choudhary, E. P. Hamilton, and D. Roth, "A comparative study of fairness-enhancing interventions in machine learning," in *Proceedings of the Conference on Fairness, Accountability, and Transparency*,

ser. FAT* '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 329–338. [Online]. Available: https://doi.org/10.1145/3287560.3287589

[11] X. Ferrer, T. v. Nuenen, J. M. Such, M. Coté, and N. Criado, "Bias and discrimination in ai: A cross-disciplinary perspective," *IEEE Technology and Society Magazine*, vol. 40, no. 2, pp. 72–80, 2021.

[12] D. Pedreshi, S. Ruggieri, and F. Turini, "Discrimination-aware data mining," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 560–568, 2008. [Online]. Available: https://dl.acm.org/doi/10.1145/1401890.1401959

[13] M. Hort and F. Sarro, "Did you do your homework? raising awareness on software fairness and discrimination," *Proceedings - 2021 36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021*, pp. 1322–1326, 2021.

[14] "Fairness in criminal justice risk assessments: The state of the art," *Sociological Methods and Research*, vol. 50, pp. 3–44, 2 2021.

[15] N. Kozodoi, J. Jacob, and S. Lessmann, "Fairness in credit scoring: Assessment, implementation and profit implications," *European Journal of Operational Research*, vol. 297, pp. 1083–1094, 3 2022.

[16] K. Albusays, P. Bjorn, L. Dabbish, D. Ford, E. Murphy-Hill, A. Serebrenik, and M.-A. Storey, "The diversity crisis in software development," *IEEE Software*, vol. 38, no. 2, pp. 19–25, 2021.

[17] Y. Li, L. Meng, L. Chen, L. Yu, D. Wu, Y. Zhou, and B. Xu, "Training data debugging for the fairness of machine learning software," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 2215–2227. [Online]. Available: https://doi.org/10.1145/3510003.3510091

[18] S. Biswas and H. Rajan, "Fair preprocessing: Towards understanding compositional fairness of data transformers in machine learning pipeline," *ESEC/FSE 2021 - Proceedings of the 29th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, vol. 21, pp. 981–993, 8 2021. [Online]. Available: https://dl.acm.org/doi/10.1145/3468264.3468536

[19] M. Hort, J. M. Zhang, F. Sarro, and M. Harman, "Fairea: A model behaviour mutation approach to benchmarking bias mitigation methods," *ESEC/FSE 2021 - Proceedings of the 29th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 994–1006, 8 2021. [Online]. Available: https://dl.acm.org/doi/10.1145/3468264.3468565

[20] J. Chakraborty, S. Majumder, and T. Menzies, "Bias in machine learning software: Why? how? what to do?" *ESEC/FSE 2021 - Proceedings of the 29th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 429–440, 8 2021. [Online]. Available: https://doi.org/10.1145/3468264.3468537

[21] J. Chakraborty, S. Majumder, Z. Yu, and T. Menzies, "Fairway: a way to build fair ml software." ACM, 11 2020, pp. 654–665. [Online]. Available: https://dl.acm.org/doi/10.1145/3368089.3409697

[22] S. Biswas and H. Rajan, "Do the machine learning models on a crowd sourced platform exhibit bias? an empirical study on model fairness," *ESEC/FSE 2020 - Proceedings of the 28th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 642–653, 11 2020.

[23] S. Tizpaz-Niari, A. Kumar, G. Tan, and A. Trivedi, "Fairness-aware configuration of machine learning libraries," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 909–920. [Online]. Available: https://doi.org/10.1145/3510003.3510202

[24] J. M. Zhang and M. Harman, "'ignorance and prejudice' in software fairness," *Proceedings - International Conference on Software Engineering*, pp. 1436–1447, 5 2021.

[25] Y. Brun and A. Meliou, "Software fairness," ser. ESEC/FSE 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 754–759. [Online]. Available: https://doi.org/10.1145/3236024.3264838

[26] E. Soremekun, M. Papadakis, M. Cordy, and Y. L. Traon, "Software fairness: An analysis and survey," 5 2022. [Online]. Available: https://arxiv.org/abs/2205.08809v1http://arxiv.org/abs/2205.08809

[27] D. Badampudi, M. Unterkalmsteiner, and R. Britto, "Modern code reviews—survey of literature and practice," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 4, may 2023. [Online]. Available: https://doi.org/10.1145/3585004

[28] H. A. Çetin, E. Doğan, and E. Tüzün, "A review of code reviewer recommendation studies: Challenges and future directions," *Science of Computer Programming*, vol. 208, p. 102652, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167642321000459

[29] P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida, and K. I. Matsumoto, "Who should review my code? a file location-based code-reviewer recommendation approach for modern code review," *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015 - Proceedings*, pp. 141–150, 4 2015.

[30] M. Zehlike, F. Bonchi, C. Castillo, S. Hajian, M. Megahed, and R. Baeza-Yates, "Fa*ir: A fair top-k ranking algorithm," *International Conference on Information and Knowledge Management, Proceedings*, vol. Part F131841, pp. 1569–1578, 11 2017. [Online]. Available: https://dl.acm.org/doi/10.1145/3132847.3132938

[31] S. C. Geyik, S. Ambler, and K. Kenthapadi, "Fairness-aware ranking in search and recommendation systems with application to linkedin talent search," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 2221–2231, 7 2019. [Online]. Available: https://dl.acm.org/doi/10.1145/3292500.3330691

[32] M. D. Ekstrand, M. Tian, I. M. Azpiazu, J. D. Ekstrand, O. Anuyah, D. McNeill, and M. S. Pera, "All the cool kids, how do they fit in?: Popularity and demographic biases in recommender evaluation and effectiveness," in *Proceedings of the 1st Conference on Fairness, Accountability and Transparency*, ser. Proceedings of Machine Learning Research, S. A. Friedler and C. Wilson, Eds., vol. 81. PMLR, 23–24 Feb 2018, pp. 172–186. [Online]. Available: https://proceedings.mlr.press/v81/ekstrand18b.html

[33] B. Rastegarpanah, K. P. Gummadi, and M. Crovella, "Fighting fire with fire: Using antidote data to improve polarization and fairness of recommender systems," in *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, ser. WSDM '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 231–239. [Online]. Available: https://doi.org/10.1145/3289600.3291002

[34] S. Tuarob, N. Assavakamhaenghan, W. Tanaphantaruk, P. Suwanworaboon, S. U. Hassan, and M. Choetkiertikul, "Automatic team recommendation for collaborative software development," *Empirical Software Engineering*, vol. 26, pp. 1–53, 7 2021, https://github.com/noppadol-assava/RECAST. [Online]. Available: https://link.springer.com/article/10.1007/s10664-021-09966-4

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

[36] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," *Proceedings - International Conference on Software Engineering*, pp. 712–721,

2013.

[37] C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli, "Modern code review: a case study at google," in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 181–190. [Online]. Available: https://doi.org/10.1145/3183519.3183525

[38] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "An empirical study of the impact of modern code review practices on software quality," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2146–2189, 2016. [Online]. Available: https://doi.org/10.1007/s10664-015-9381-9

[39] C. Thompson and D. Wagner, "A large-scale study of modern code review and security in open source projects," in *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*, ser. PROMISE. New York, NY, USA: Association for Computing Machinery, 2017, p. 83–92. [Online]. Available: https://doi.org/10.1145/3127005.3127014

[40] L. Nurgalieva, A. Frik, and G. Doherty, "A narrative review of factors affecting the implementation of privacy and security practices in software development," *ACM Comput. Surv.*, vol. 55, no. 14s, jul 2023. [Online]. Available: https://doi.org/10.1145/3589951

[41] N. Davila and I. Nunes, "A systematic literature review and taxonomy of modern code review," *Journal of Systems and Software*, vol. 177, p. 110951, 7 2021.

[42] C. Bird, T. Carnahan, and M. Greiler, "Lessons learned from building and deploying a code review analytics platform," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, ser. MSR '15. IEEE Press, 2015, p. 191–201.

[43] G. Kudrjavets, A. Kumar, N. Nagappan, and A. Rastogi, "Mining code review data to understand waiting times between acceptance and merging: an empirical analysis," in *Proceedings of the 19th International Conference on Mining Software Repositories*, ser. MSR '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 579–590. [Online]. Available: https://doi.org/10.1145/3524842.3528432

[44] A. Bosu, J. C. Carver, C. Bird, J. Orbeck, and C. Chockley, "Process aspects and social dynamics of contemporary code review: Insights from open source development and industrial practice at microsoft," *IEEE Transactions on Software Engineering*, vol. 43, no. 1, pp. 56–75, 2017.

[45] P. C. Rigby and C. Bird, "Convergent contemporary software peer review practices," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2013. New York, NY, USA: Association for Computing Machinery, 2013, p. 202–212. [Online]. Available: https://doi.org/10.1145/2491411.2491444

[46] G. Jeong, S. Kim, T. Zimmermann, and K. Yi, "Improving code review by predicting reviewers and acceptance of patches," *Research on software analysis for error-free computing center Tech-Memo (ROSAEC MEMO 2009-006)*, pp. 1–18, 2009.

[47] 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.

[48] E. Sülün, E. Tüzün, and U. Doğrusöz, "Rstrace+: Reviewer suggestion using software artifact traceability graphs," *Information and Software Technology*, vol. 130, p. 106455, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584920300021

[49] Z. Xia, H. Sun, J. Jiang, X. Wang, and X. Liu, "A hybrid approach to code reviewer recommendation with collaborative filtering," in *2017 6th International Workshop on Software Mining (SoftwareMining)*, 2017, pp. 24–31.

[50] A. Strand, M. Gunnarson, R. Britto, and M. Usman, "Using a context-aware approach to recommend code reviewers: Findings from an industrial case study," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1–10. [Online]. Available: https://doi.org/10.1145/3377813.3381365

[51] A. Chueshev, J. Lawall, R. Bendraou, and T. Ziadi, "Expanding the number of reviewers in open-source projects by recommending appropriate developers," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2020, pp. 499–510.

[52] "How do fairness definitions fare? examining public attitudes towards algorithmic definitions of fairness," *AIES 2019 - Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pp. 99–106, 1 2019. [Online]. Available: https://dl.acm.org/doi/10.1145/3306618.3314248

[53] D. Kaur, S. Uslu, K. J. Rittichier, and A. Durresi, "Trustworthy artificial intelligence: A review," *ACM Computing Surveys*, vol. 55, 3 2023.

[54] M. Kearns, S. Neel, A. Roth, and Z. S. Wu, "An empirical study of rich subgroup fairness for machine learning," in *Proceedings of the Conference on Fairness, Accountability, and Transparency*, ser. FAT* '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 100–109. [Online]. Available: https://doi.org/10.1145/3287560.3287592

[55] ——, "Preventing fairness gerrymandering: Auditing and learning for subgroup fairness," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 2564–2572. [Online]. Available: https://proceedings.mlr.press/v80/kearns18a.html

[56] Y. Wang, W. Ma, M. Zhang, Y. Liu, and S. Ma, "A survey on the fairness of recommender systems," *ACM Transactions on Information Systems*, vol. 41, pp. 1–43, 7 2023. [Online]. Available: https://doi.org/10.1145/3547333

[57] J. Chen, H. Dong, X. Wang, F. Feng, M. Wang, H. Dong, X. Wang, F. Feng, and X. He, "Bias and debias in recommender system: A survey and future directions," *ACM Transactions on Information Systems*, vol. 41, p. 67, 2 2023. [Online]. Available: https://dl.acm.org/doi/10.1145/3564284

[58] A. Beutel, J. Chen, T. Doshi, H. Qian, L. Wei, Y. Wu, L. Heldt, Z. Zhao, L. Hong, E. H. Chi, and C. Goodrow, "Fairness in recommendation ranking through pairwise comparisons," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery; Data Mining*, ser. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 2212–2220. [Online]. Available: https://doi.org/10.1145/3292500.3330745

[59] Y. Li, H. Chen, S. Xu, Y. Ge, and Y. Zhang, "Towards personalized fairness based on causal notion," in *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1054–1063. [Online]. Available: https://doi.org/10.1145/3404835.3462966

[60] M. Kaya, D. Bridge, and N. Tintarev, "Ensuring fairness in group recommendations by rank-sensitive balancing of relevance," in *Proceedings of the 14th ACM Conference on Recommender Systems*, ser. RecSys '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 101–110. [Online]. Available: https://doi.org/10.1145/3383313.3412232

[61] M. Morik, A. Singh, J. Hong, and T. Joachims, "Controlling fairness and bias in dynamic learning-to-rank," in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 429–438. [Online]. Available: https://doi.org/10.1145/3397271.3401100

[62] J. Kleinberg, "Inherent trade-offs in algorithmic fairness," *SIGMETRICS Perform. Eval. Rev.*, vol. 46, no. 1, p. 40, jun 2018. [Online]. Available: https://doi.org/10.1145/3292040.3219634

[63] S. Corbett-Davies, E. Pierson, A. Feller, S. Goel, and A. Huq, "Algorithmic decision making and the cost of fairness," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 797–806. [Online]. Available: https://doi.org/10.1145/3097983.3098095

[64] Z. C. Lipton, A. Chouldechova, and J. McAuley, "Does mitigating ml's impact disparity require treatment disparity?" in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 8136–8146.

[65] Y. Bechavod and K. Ligett, "Penalizing unfairness in binary classification," *arXiv preprint arXiv:1707.00044*, 2017.

[66] S. A. Friedler, C. Scheidegger, S. Venkatasubramanian, S. Choudhary, E. P. Hamilton, and D. Roth, "A comparative study of fairness-enhancing interventions in machine learning," in *Proceedings of the Conference on Fairness, Accountability, and Transparency*, ser. FAT* '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 329–338. [Online]. Available: https://doi.org/10.1145/3287560.3287589

[67] A. K. Menon and R. C. Williamson, "The cost of fairness in binary classification," in *Proceedings of the 1st Conference on Fairness, Accountability and Transparency*, ser. Proceedings of Machine Learning Research, S. A. Friedler and C. Wilson, Eds., vol. 81. PMLR, 23–24 Feb 2018, pp. 107–118. [Online]. Available: https://proceedings.mlr.press/v81/menon18a.html

[68] A. Albarghouthi, L. D'Antoni, S. Drews, and A. V. Nori, "Fairsquare: Probabilistic verification of program fairness," *Proc. ACM Program. Lang.*, vol. 1, no. OOPSLA, oct 2017. [Online]. Available: https://doi.org/10.1145/3133904

[69] D. M. German, G. Robles, G. Poo-Caamaño, X. Yang, H. Iida, and K. Inoue, "'"was my contribution fairly reviewed?"': A framework to study the perception of fairness in modern code reviews," in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 523–534. [Online]. Available: https://doi.org/10.1145/3180155.3180217

[70] C. Ferrara, G. Sellitto, F. Ferrucci, F. Palomba, and A. D. Lucia, "Fairness-aware machine learning engineering: how far are we?" *Empirical Software Engineering*,

vol. 29, pp. 1–46, 2 2024. [Online]. Available: https://link.springer.com/article/10.1007/s10664-023-10402-y

[71] S. Sultana, A. K. Turzo, and A. Bosu, "Code reviews in open source projects: How do gender biases affect participation and outcomes?" *Empirical Softw. Engg.*, vol. 28, no. 4, jun 2023. [Online]. Available: https://doi.org/10.1007/s10664-023-10324-9

[72] K. A. Tecimer, E. Tüzün, H. Dibeklioglu, and H. Erdogmus, "Detection and elimination of systematic labeling bias in code reviewer recommendation systems," in *Evaluation and Assessment in Software Engineering*, ser. EASE 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 181–190. [Online]. Available: https://doi.org/10.1145/3463274.3463336

[73] E. Murphy-Hill, J. Dicker, A. Horvath, M. M. Hodges, C. D. Egelman, L. R. Weingart, C. Jaspan, C. Green, and N. Chen, "Systemic gender inequities in who reviews code," *Proc. ACM Hum.-Comput. Interact.*, vol. 7, no. CSCW1, apr 2023. [Online]. Available: https://doi.org/10.1145/3579527

[74] P. Emerson, "The original borda count and partial voting," *Social Choice and Welfare*, vol. 40, no. 2, pp. 353–358, 2013.

[75] Y. Li, H. Chen, Z. Fu, Y. Ge, and Y. Zhang, "User-oriented fairness in recommendation," *The Web Conference 2021 - Proceedings of the World Wide Web Conference, WWW 2021*, pp. 624–632, 4 2021. [Online]. Available: https://dl.acm.org/doi/10.1145/3442381.3449866

[76] A. J. Biega, K. P. Gummadi, and G. Weikum, "Equity of attention: Amortizing individual fairness in rankings," *41st International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2018*, vol. 18, pp. 405–414, 6 2018. [Online]. Available: https://dl.acm.org/doi/10.1145/3209978.3210063

[77] M. Naghiaei, H. A. Rahmani, and Y. Deldjoo, "Cpfair: Personalized consumer and producer fairness re-ranking for recommender systems," in *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 770–779. [Online]. Available: https://doi.org/10.1145/3477495.3531959

[78] G. K. Patro, A. Biswas, N. Ganguly, K. P. Gummadi, and A. Chakraborty, "Fairrec: Two-sided fairness for personalized recommendations in two-sided platforms," in *Proceedings of The Web Conference 2020*, ser. WWW '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1194–1204. [Online]. Available: https://doi.org/10.1145/3366423.3380196

[79] Z. Zhu, Y. He, X. Zhao, Y. Zhang, J. Wang, and J. Caverlee, "Popularity-opportunity bias in collaborative filtering," in *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, ser. WSDM '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 85–93. [Online]. Available: https://doi.org/10.1145/3437963.3441820

[80] A. Ferraro, "Music cold-start and long-tail recommendation: Bias in deep representations," in *Proceedings of the 13th ACM Conference on Recommender Systems*, ser. RecSys '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 586–590. [Online]. Available: https://doi.org/10.1145/3298689.3347052

[81] S. Liu and Y. Zheng, "Long-tail session-based recommendation," in *Proceedings of the 14th ACM Conference on Recommender Systems*, ser. RecSys '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 509–514. [Online]. Available: https://doi.org/10.1145/3383313.3412222

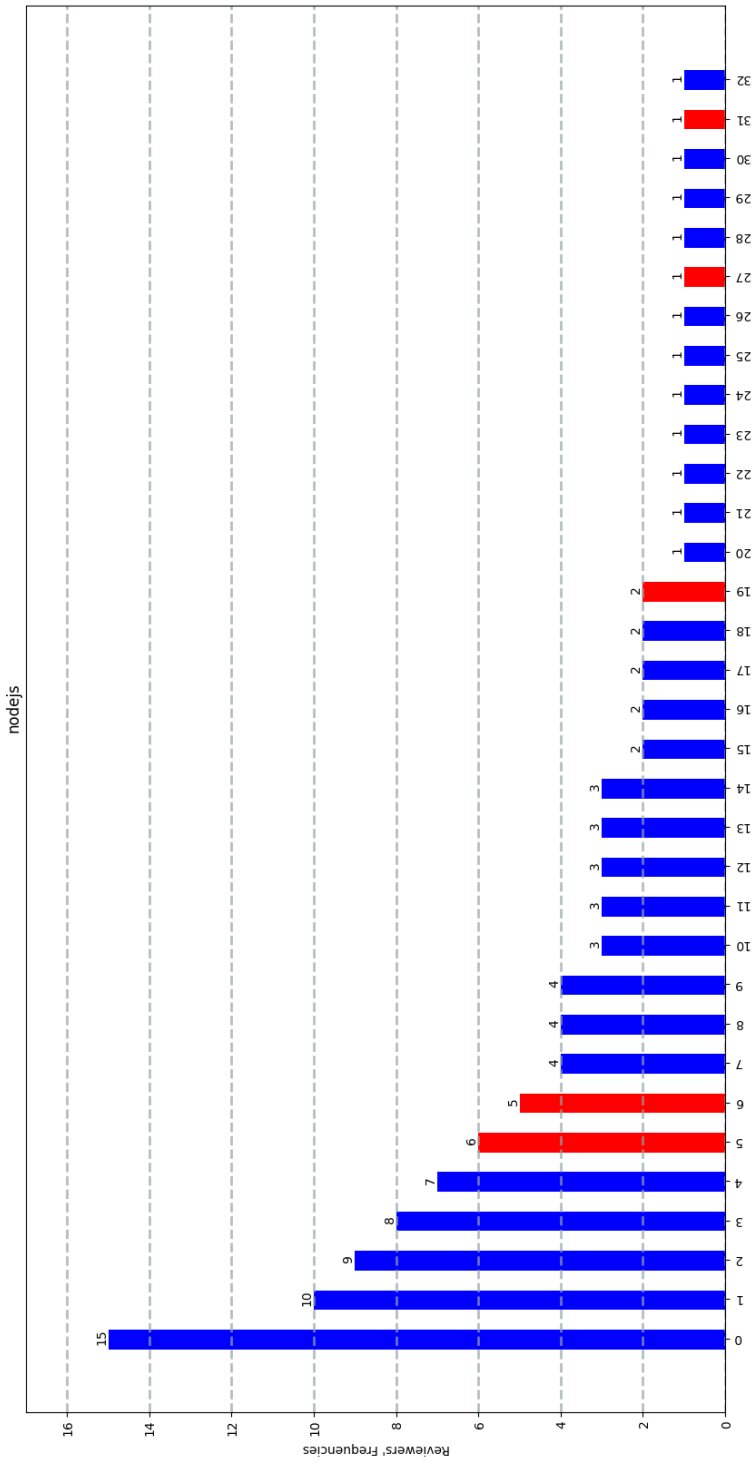
[82] R. F. Woolson, *Wilcoxon Signed-Rank Test*. John Wiley and Sons, Ltd, 2008, pp. 1–3. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780471462422.eoct979

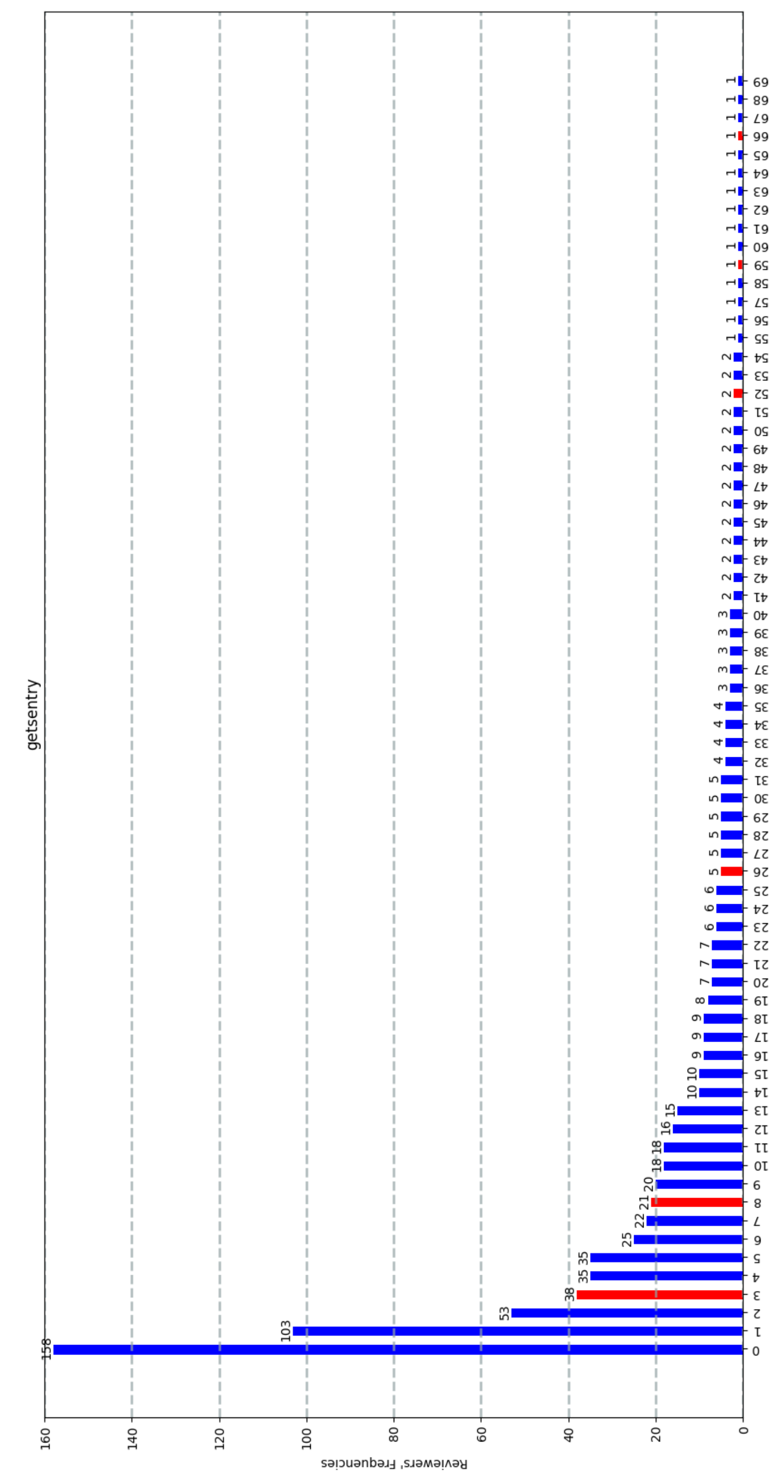
Appendix A

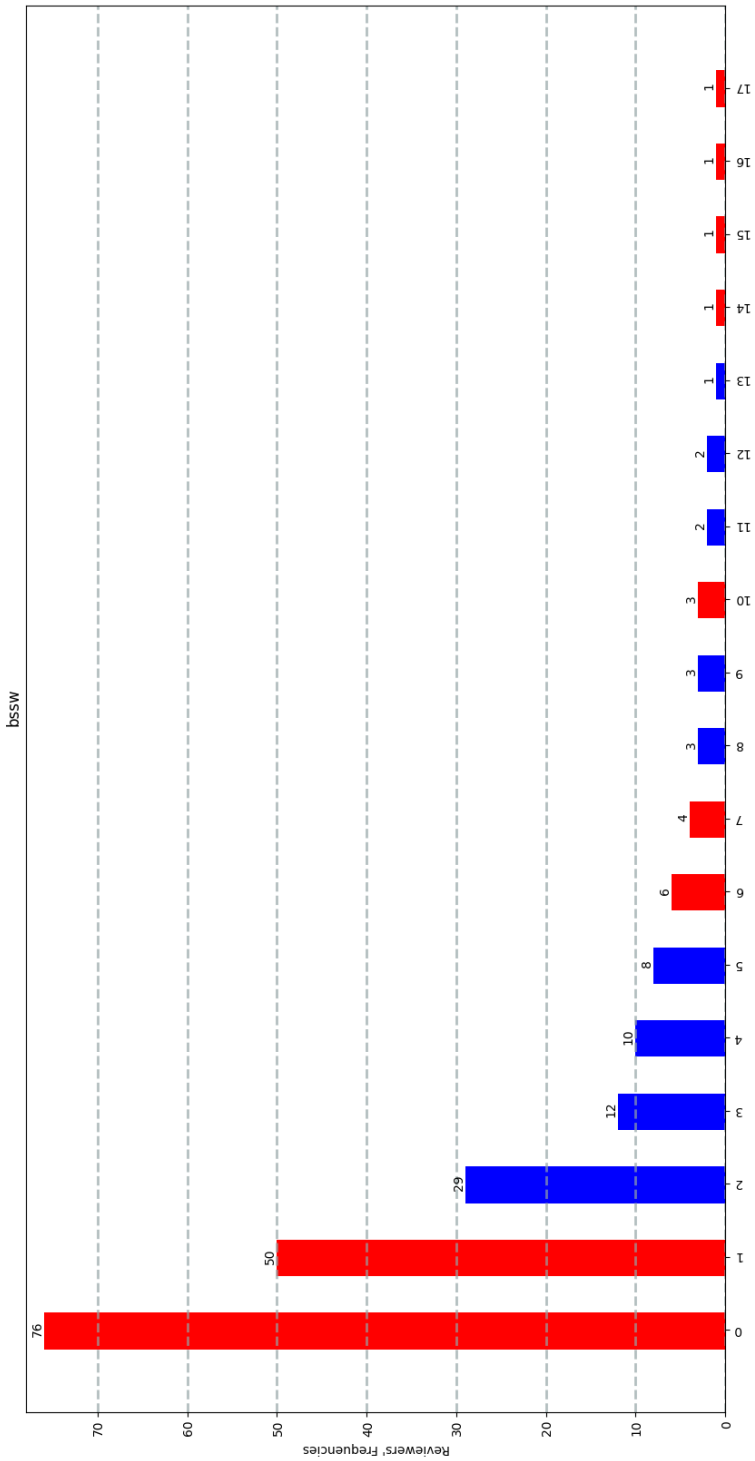
Supplementary Materials

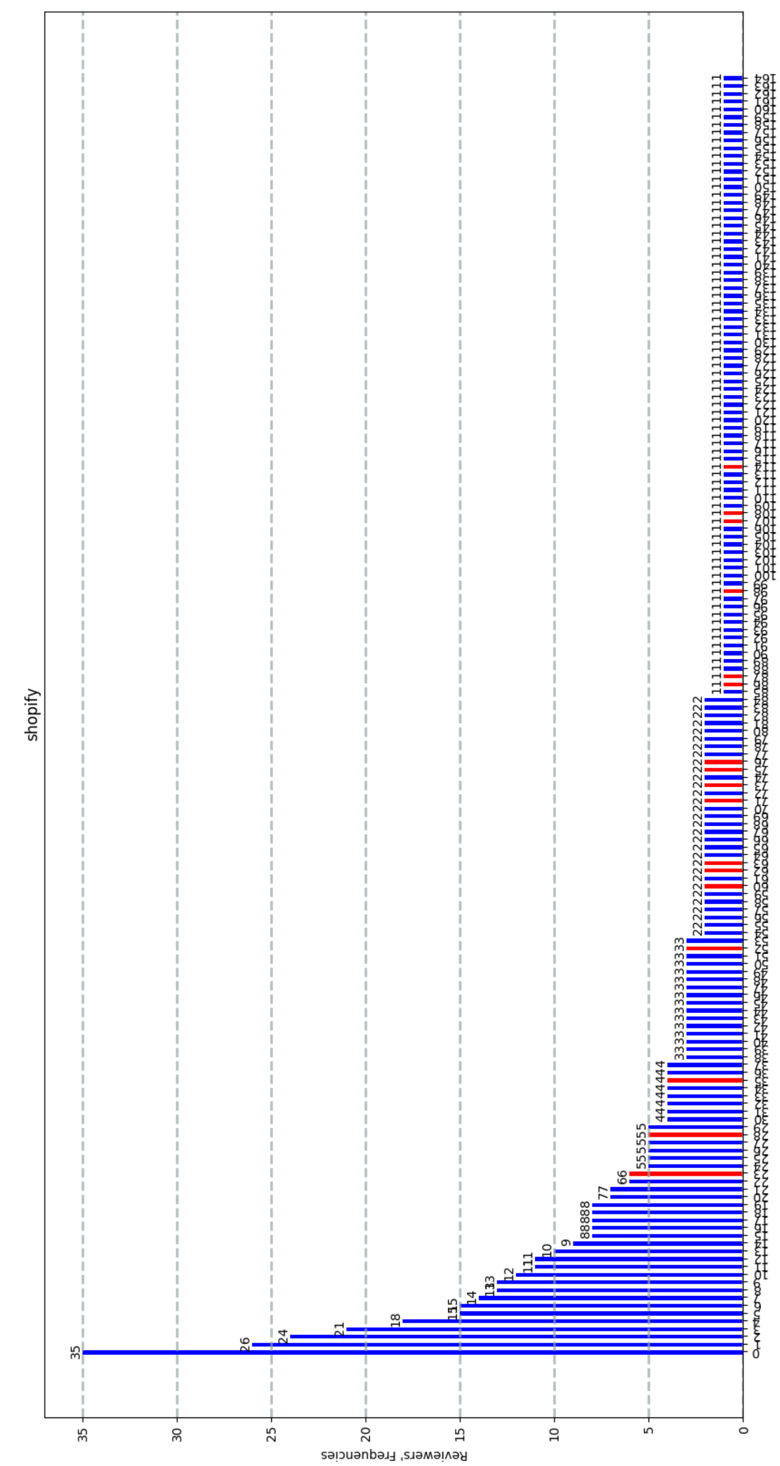
A.1 Long Tail Charts for Reviewers' Genders Frequency

In these charts, the blue bars are male reviewers, and the red ones are female reviewers.


nodejs
Reviewers' frequencies
0 15
1 10
2 9
3 8
4 7
5 6
6 5
7 4
8 4
9 4
10 3
11 3
12 3
13 3
14 3
15 2
16 2
17 2
18 2
19 2
20 1
21 1
22 1
23 1
24 1
25 1
26 1
27 1
28 1
29 1
30 1
31 1
32 1
0 2 4 6 8 10 12 14 16


getsentry
Reviewers' Frequencies


bssw
Reviewers' frequencies
0
10
20
30
40
50
60
70
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
76
50
29
12
10
8
9
4
3
3
3
2
2
1
1
1
1
1


shopify
Reviewers' Frequencies

Appendix B

Data Availability

We release the dataset and source code of our experiments to help other researchers replicate and extend our study[2].

[2] https://doi.org/10.5281/zenodo.7897538