Investigating the Validity of Ground Truth in Code Reviewer Recommendation Studies

Emre Doğan
Department of Computer Engineering
Bilkent University
Ankara, Turkey
emre.dogan@bilkent.edu.tr

Eray Tüzün
Department of Computer Engineering
Bilkent University
Ankara, Turkey
eraytuzun@cs.bilkent.edu.tr

K. Ayberk Tecimer
Department of Computer Engineering
Bilkent University
Ankara, Turkey
ayberk.tecimer@ug.bilkent.edu.tr

H. Altay Güvenir
Department of Computer Engineering
Bilkent University
Ankara, Turkey
guvenir@cs.bilkent.edu.tr

Abstract:

Background: Selecting the ideal code reviewer in modern code review is a crucial first step to perform effective code reviews. There are several algorithms proposed in the literature for recommending the ideal code reviewer for a given pull request. The success of these code reviewer recommendation algorithms is measured by comparing the recommended reviewers with the ground truth that is the assigned reviewers selected in real life. However, in practice, the assigned reviewer may not be the ideal reviewer for a given pull request.
Aims: In this study, we investigate the validity of ground truth data in code reviewer recommendation studies.
Method: By conducting an informal literature review, we compared the reviewer selection heuristics in real life and the algorithms used in recommendation models. We further support our claims by using empirical data from code reviewer recommendation studies.
Results: By literature review, and accompanying empirical data, we show that ground truth data used in code reviewer recommendation studies is potentially problematic. This reduces the validity of the code reviewer datasets and the reviewer recommendation studies.
Conclusion: We demonstrated the cases where the ground truth in code reviewer recommendation studies are invalid and discussed the potential solutions to address this issue.
Keywords— reviewer recommendation, ground truth, cognitive bias, attribute substitution, systematic noise, threats to validity.

I. Introduction

Code review is an important task in software development life cycle. It is the examination of a changeset or a pull request in order to detect all possible problems before merging the change into the main branch.

There have been several studies regarding the code review process for more than 40 years. In 1976, Michael Fagan coined the term *code inspection*, which is known to be the first systematic code review methodology to reduce errors in software development process. This method is also known as *Fagan Inspection* and is based on the review efficiency at IBM by setting up inspection meetings [1]. In the last few decades, software projects have progressed in terms of the number of authors and the lines of code. This raises the necessity of more practical and more flexible methods as known as modern code review.

As the size of software development teams has increased, assigning an appropriate reviewer for a code changeset became an essential task. A software company which releases a new version of a product needs the code to be checked in the shortest possible time. In order to speed up and optimize the reviewer assignment process, different studies have been proposed in the literature. The main focus of these studies is to predict the ideal reviewer for a pull request or a changeset. Automation of the reviewer assignment process can also save a considerable amount of time and effort in larger projects.

There are a large number of code reviewer recommendation studies using a variety of machine learning algorithms [2]–[9]. The structure of these models can be summarized with the famous definition of Tom M. Mitchell for machine learning [10]:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."

In the case of code reviewer recommendation, the task is to recommend the ideal reviewer for a given pull request. The performance measure is usually chosen from among some different metrics such as *mean reciprocal rank (MRR), top-k accuracy, precision* or *recall*. The experience consists of the previous reviewer assignments collected from open source and proprietary software projects. These assignments are assumed to be ideal and new reviewers are selected with respect to the experience learnt from the previous assignments. In this paper, we investigate the ground truth of reviewer assignments collected from real-life scenarios. We claim that the performance evaluation of the reviewer recommendation studies using real-life data is potentially questionable in terms of the validity. Within our research, we define the following research questions:

RQ1: ***What kind of methods are used in code reviewer recommendation studies?***

RQ2: ***What are the factors affecting the code reviewer selection process in real life?***

The reviewer recommendation models evaluate their accuracy based on real-life data. In these models, a code reviewer assigned in an open-source or proprietary software project is assumed to be an ideal reviewer. In this study, we define the ideal reviewer as the theoretical best possible reviewer in the team that would improve or preferably perfect (such as pointing out all the defects) the pull request under review. The selection of the ideal reviewer is assumed to be performed by considering only the technical factors and ignoring any kind of non-technical factors such as availability of the reviewer.

However, some case studies completed in real-life software projects show that reviewer assignments are highly affected by several human factors such as the most appropriate reviewer not being available for a review when needed [11], or assigning a volunteer as the reviewer or assigning a close friend as the reviewer [12]. These factors show that the reviewer assignments in real life are not always done with the aim of necessarily finding the ideal reviewer. The inconsistency between the answers of RQ1 and RQ2 lead us to address a new research question:

978-1-7281-2968-6/19/$31.00 ©2019 IEEE

RQ3: ***How reliable is the ground truth in code reviewer recommendation studies?***

In this paper, we investigate the validity of ground truth in the evaluation of the code reviewer recommendation studies. To the best of our knowledge, the ground truth problem for the code reviewer recommendation task has not been investigated by any other study in the literature.

In the following section, we present the background information related to code review and ground truth. In Section III, we expand our problem definition in detail. In Section IV, we discuss the answers to the research questions. In Section V, we propose potential solutions to overcome the described ground truth problem. Finally, Section VI presents our concluding remarks and future directions.

II. Background

A. Modern Code Review

Code review is the manual assessment of source code by developers in order to identify defects and quality problems [13]. Fagan style traditional code review is a formal and well-documented code review process by inspecting the code in meetings. These meetings were helpful and showed a successful defect detection efficiency [14]. However, its time and organization related costs are higher than the non-meeting-based review methods [15]. In order to solve this problem, a more light-weight and tool-based code review process known as modern code review has become popular among many open-source and proprietary projects [16]. According to the survey completed in 2017, among 240 software development teams, 90% of teams prefer to use modern code review approach, also known as change-based code review [17].

B. Code Reviewer Recommendation Studies

With the growing complexity of software projects over the years, the necessity of automating code reviewer assignments has increased. In the literature, different approaches have been proposed to find the ideal developer for a code review. Lipčák and Rossi [5] categorized the existing recommendation approaches into four groups: heuristic approaches, machine learning approaches, social networks and hybrid approaches.

Heuristic models follow a similar approach as in real life to find the ideal code reviewer. Some meaningful metrics from real life such as *path similarity* [5], [18]–[22], *expertise in the related project and technologies* [23]–[25] and *previous review success* [26] are used to automate the assignment process.

With the current improvements in the machine learning field, recommendation systems in software engineering have shown significant progress. In [3]–[5], [8], Bayesian networks are used to predict the ideal reviewer. There are also a few studies using support vector machine [3], [7] and random forest classifier [2], [3] for the recommendation task. Ouni et al. [6] uses a genetic algorithm approach to find the ideal reviewer.

There are also some graph-based recommendation approaches available in the literature. Lee et al. [31] investigates the links between developers and proposes a recommendation system based on these links. Similarly in [21], [28]–[30], the effect of social network relations is used for code reviewer recommendation.

TABLE I. STUDIES PROPOSING AN AUTOMATED REVIEWER RECOMMENDATION MODEL

RELATED STUDIES	APPLIED METHOD
[3]–[5], [8]	Bayesian Network
[25]	Change History of Source Code
[3]	Decision Tree
[23], [24]	Expertise in Related Technologies
[6]	Genetic Algorithm
[20]	Information Retrieval
[3]	K-nearest neighbors (KNN)
[27]	Latent Dirichlet Allocation
[9]	Latent Factor & Neighborhood
[5], [18]–[22]	Path Similarity
[26]	Previous Review Success
[2], [3]	Random Forest
[21], [28]–[32]	Graph-based Analysis
[3], [7]	Support Vector Machines
[21]	Text Similarity of PR Requests

These approaches can be combined to perform better. For example, CoreDevRec extracts some features in a heuristic manner and uses them for training an SVM (Support Vector Machine) for the recommendation task [7].

A detailed list of code reviewer recommendation studies with their approaches is given in Table I.

Although there is not an explicit study investigating the problematic ground truth in the code reviewer recommendation domain, a few studies mention our concern of the problematic ground truth in threats to validity section without offering any kind of mitigations:

"*We do not know that these reviewers were the best nor other reviewers were equally capable (but did not contribute due to issues such as workload and schedule).*" [26]

"…, we use the real reviewers of pull requests as the true result, however, the true reviewers may not be the best reviewers. So our recommendation has the risk of recommending reviewers that are not the best suitable." [9]

C. Ground Truth Problems

Ground truth can be defined as the ideal output label of an algorithm. It provides a reference point for the evaluation of the algorithm. There are some cases in which the ground truth is not defined correctly and is different from the output label. Using data with such a problematic ground truth definition may lead to invalid results.

Although there is not a consensus on the definition of this problem, many studies investigate the problematic ground truth as noise in data. In terms of its characteristics, there are two types of noise in data:

- *Data with Random Noise:* This type of data suffers from noise without a discernible pattern which means that the labels of data are attached in a faulty way randomly.
- *Data with Systematic Noise:* Unlike the random noise, there is a particular noise pattern in this type of data.

The data instances with systematic noise can cause ground truth to be biased in data.

Bias in ground truth data is a critical issue that needs to be clarified in software engineering related studies. According to the recent study by Mohanini et al. [33], an important cause of software project challenges and failures is the systematic errors introduced by human cognitive biases. Bird et al. [34] investigate the links between bug reports and the source code, and find that there exists a systematic bias in the severity levels of defects and the experience level of developers. The authors argue that this systematic bias threatens the validity of prediction models and the generalizability of hypotheses tested on biased data.

III. Problem Definition

A common scenario for a code reviewer assignment follows the pattern listed below:

1. A developer (author of the pull request) initiates a pull request in order to either fix a problem or implement a feature.
2. The team leader assigns reviewer(s) to the pull request considering all available reviewers. (In some settings, the developer may also propose reviewer(s)).
3. The assigned reviewer provides some comments and gives necessary feedback on the pull request.
4. The developer updates the pull request by making the necessary changes.
5. The loop between the reviewer and the developer continues until they come to an agreement.
6. The pull request is merged to the main line.

All of the code reviewer recommendation studies in the literature use datasets consisting of the reviewer assignments that are collected from previously completed real-life projects. These studies assume that the team leader always assigns the ideal reviewer in Step 2. However, the ideal reviewer might not be assigned due to several reasons:

- *Availability Reasons:* Here, the team leader identifies the ideal reviewer correctly, however the assignment has to be changed for some reasons such as:
  - The ideal reviewer might be physically absent from work, so he/she cannot review the pull request.
  - The ideal reviewer might be busy with some other tasks, so he/she declines to review the pull request.
  - The ideal reviewer might be busy with some other tasks and is late to reply the review request, so the team leader has to assign someone else.

- *Cognitive Biases:* Selecting the appropriate code reviewer is a decision-making activity involving human decision makers, thus open to cognitive biases. Here, cognitive biases are defined as systematic deviations from optimal reasoning [33]. One of the cognitive biases is attribute substitution. Attribute substitution occurs when an individual has to make a judgment (of a target attribute) that is computationally complex, and instead substitutes a more easily calculated heuristic attribute. Here, instead of trying to find the ideal reviewer for a given pull request (which might be time consuming and not a straightforward task), a team leader might substitute this challenging task with easier heuristics such as the following:
  - The team leader prefers to assign a volunteer for the review [11], [35].
  - The team leader prefers to assign a reviewer based on their work schedule [11].
  - The team leader prefers to assign a new hire as a reviewer for educational purposes [16].
  - The team leader might prefer to assign a developer based on their relative response time to review request [11].

In either of these cases, the reasons for assigning the code reviewer might be different than the conditions considered by a recommendation technique. We argue that this discrepancy creates a potential problem in the ground truth.

IV. Research Questions

In the following subsections (IV.A and IV.B), code reviewer selection process by both manual and automated methods are described. In Section IV.C, the selection of code reviewers in real life and the reviewer recommendation algorithms are compared while questioning the validity of the ground truth in code reviewer recommendation studies. We further provided quantitative evidence for the ground truth problem.

A. RQ1 – Reviewer Selection in Recommendation Systems

As illustrated in Table I, there are different solution approaches for the code reviewer recommendation task in the literature. Some of these approaches propose metrics to recommend the ideal developer (*i.e. change history of code lines* [25]*, path similarity* [5], [18]–[22]). Other studies recommend reviewers by using different machine learning algorithms [2]–[9] and graph based approaches [21], [28]–[31].

The aim of these algorithms is to recommend the ideal reviewer for a given pull request by learning from previous reviewer assignments.

B. RQ2 – Reviewer Selection in Real Life

There are different case studies investigating the human factors that affect code review assignments. Bird et al. [11] state that beyond the technical aspects (code familiarity, review experience etc.), there are some social and personal factors (availability, response time to the review request etc.) affecting the attendance ratio and quality of a code review.

TABLE II. HUMAN FACTORS AFFECTING HOW REVIEWERS ARE SELECTED

RELATED STUDIES	FACTORS
[11], [36], [37]	Review experience
[11], [12], [35]	Code familiarity
[36], [37]	Patch characteristics
[11], [35]	**Volunteer for review**
[11], [36]	**Workload**
[11]	**Physical proximity**
[11]	**Availability**
[11]	**Response time to review requests**
[35]	**Training the new hires**
[12]	**Social interactions**

TABLE III. AN EXTENDED SET OF REAL-LIFE REVIEWER ASSIGNMENT SCENARIOS

Pull Request	Assigned Reviewer	Ideal Reviewer	Recommended Reviewer	Case	Correctness of Algorithm w.r.t. Assigned Reviewer	Correctness of Algorithm w.r.t. Ideal Reviewer	Ground Truth Validity
1	John	John	John	$r = a \ and \ a = i$	✓	✓	Yes
2	John	John	Mary	$r \neq a \ and \ a = i$	X	X	Yes
3	**John**	**Mary**	**John**	$\boldsymbol{r = a \ and \ a \neq i}$	✓	X	**No**
4	**John**	**Mary**	**Mary**	$\boldsymbol{r \neq a \ and \ a \neq i}$	X	✓	**No**
5	**John**	**Mary**	**James**	$\boldsymbol{r \neq a \ and \ a \neq i}$	X	X	**No**

r: the recommended reviewer by the algorithm
a: the assigned reviewer in real life
i: the ideal reviewer for a pull request

Greiler et al. [35] complete a comprehensive study among Microsoft developers in order to find the critical aspects while deciding the reviewer. They find out that assigning a volunteer reviewer might be preferred rather than assigning an unwilling reviewer. Ruangwan et al. [36] emphasizes the effect of the developer workload and patch characteristics on the reviewer selection process.

The summary of the real-life factors affecting the reviewer selection is given in Table II. The factors that are in boldface might potentially create a ground truth problem as they are not technical factors aiming to find the ideal reviewer.

C. RQ3 – Comparison of Recommendation Systems and Real Life

In a real-life scenario, the assigned reviewer is not always the ideal reviewer. Although some case studies show that metrics such as code familiarity and review experience are considered in real life, most reviewer assignments are done by taking account of different human aspects. Code reviewer recommendation studies do not consider any kind of human aspects and assume that the ideal reviewer is the same as the assigned reviewer in real life. Nevertheless, in some studies [11], [12], [35], [36], it is clearly illustrated that the real-life decisions are not usually made with the goal of finding the ideal reviewer.

Based on Section IV.A and Section IV.B, it can be observed that there exists a potentially conflicting scenario for selecting code reviewers based on manual methods from real life and automated recommendation methods. This situation is prone to create a ground truth problem for the reviewer recommendation models.

To better differentiate between the problematic and unproblematic reviewer assignments, an extended version of real-life scenarios is available in Table III. This table represents the code reviewer assignments of a developer group consisting of 3 developers: *John*, *James* and *Mary*. In the interest of simplicity and clarity, all pull requests are assumed to be assigned to John as the reviewer. Although it is not an easy task to verify whether a reviewer is ideal, we assume that the ideal reviewers for each pull request in Table III are known. Each type of scenario can be explored in the following way:

In Pull Request 1, John is the assigned reviewer by the team leader and the recommended reviewer by the algorithm. It is also known that he is the ideal reviewer for this pull request. This scenario represents the $r = a \ and \ a = i$ case. There is not any ground truth problem in this case.

In Pull Request 2, the real-life reviewer is the ideal one. However, the algorithm cannot mimic neither the real-life assignment nor the ideal assignment. Although this scenario ($r \neq a \ and \ a = i$) represents an incorrect recommendation, the real-life assignments do not have a problematic ground truth.

In Pull Request 3, John is the assigned and recommended reviewer, but he is not the ideal reviewer to review this pull request. The recommendation models do not consider whether the assigned reviewer is the ideal one or not and, they consider the assigned reviewer as the ground truth. This scenario maps to the case $r = a \ and \ a \neq i$ indicating the contradiction between the assigned reviewer in real life and the ideal reviewer for a pull request.

In Pull Request 4, the algorithm recommends the ideal reviewer. However, the reasoning behind the recommendation is doubtful since the algorithm uses assigned reviewer data which is incorrect.

In Pull Request 5, it is not possible to do a reasonable evaluation as the assigned, ideal and recommended reviewers are completely different ($r \neq a \ and \ a \neq i$).

Among these scenarios, Pull Requests 3, 4 and 5 support our claim on the inconsistency between real-life and ideal reviewer assignments. We claim that these pull requests have invalid ground truth instances.

In order to provide further proof for problematic ground truth instances, an analysis of pull request reviews from four large-size software projects completed by Ruangwan et al. [36] is provided. Through a case study of 230,090 patches spread across the Android, LibreOffice, OpenStack and Qt repositories, Ruangwan et al. reveal that a remarkable number of patches (pull requests) have at least one non-responsive reviewer to the review request. The distribution of non-responsive reviewers for each project is given in Table IV.

TABLE IV. RATIOS OF THE PULL REQUESTS WITH AT LEAST ONE NON-RESPONSIVE CODE REVIEWER [36]

Project Name	Total Number of Pull Requests	Number of PRs with at least one non-responsive reviewer	The ratio of PRs having at least one non-responsive reviewer
Android	36,771	24,367	66%
LibreOffice	18,716	3,039	16%
Open Stack	108,788	24,589	23%
Qt	65,815	30,630	47%
TOTAL	230,090	82,625	36%

Table IV indicates that 82,625 pull requests of 230,090 (36%) have at least one non-responsive assigned reviewer. These numbers support our claim on the ground truth problem of real-life reviewer assignments. The assigned reviewers considered as the ideal reviewer by the team leader might not always respond to the review requests. Instead of them, some other non-ideal reviewers might be assigned for the pull request. The ratio of non-responsive reviewers in four large open-source projects varies between 16% and 66%. These statistics suggest that real-life code reviewer assignments may suffer from the *availability reasons*.

V. Solution Alternatives

As mentioned in Section III and Section IV, using the datasets collected from real-life practices to build code reviewer recommendation models is not entirely correct as the assigned code reviewer in real life may not be the ideal code reviewer. Therefore, evaluating the accuracy of the proposed algorithms with these datasets are potentially problematic. To mitigate this problem, there can be two alternative solutions for establishing valid ground truth data.

A. Creating Ground Truth Data through Experimental Setup

In this approach, all of the possible code reviewers are assigned to the same pull request as a single reviewer (i.e. each reviewer gives feedback without the knowledge of other reviewers that are also assigned to the same pull request). After each reviewer provides review feedback, ideal reviewer can be selected among those reviewers. Furthermore, it can also be used to measure the validity of ground truth dataset by comparing the assigned reviewers in the dataset with the ideal reviewers according to experiment. To explain this experiment, the steps are outlined as follows:

i. To fix an issue, developer implements a solution and commits this solution as a pull request.
ii. Team leader assigns every possible reviewer in the team as a single reviewer to the pull request.
iii. Each reviewer reviews the request and provides comments.
iv. Team leader or the developer selects the ideal review and reviewer based on expert judgement.

This selected ideal reviewer becomes the verified label for that instance. Although this approach is theoretically possible and can be used to create a valid ground truth dataset, it is expensive and impractical as each reviewer invests time to review the same pull request.

B. Cleaning Ground Truth Data

Current code reviewer recommendation algorithms use previous data (pre pull request) to predict reviewer for a given pull request. However, as we have discussed earlier, constructing reviewer recommendation model with this problematic dataset is prone to error and reduces the validity of code reviewer selection process. One way to deal with this problem is to eliminate problematic (non-ideal) reviewer labels to clean up the data.

We can improve the accuracy of the ground truth by having the assignment data checked by the actual development team from which the data is collected and ask them to correct any faulty (i.e. not ideal assignment) that may exist in the data.

Another alternative is without any human intervention, we might apply data mining algorithms or machine learning techniques to future data (post-pull request) in order to eliminate problematic reviewer labels automatically. To elaborate, consider the following example; to fix a bug, a developer creates a pull request, the assigned reviewer in the dataset approves the pull request and bug is closed. If this bug is reopened later, it is a potential indicator that the assigned reviewer was not the ideal reviewer for that pull request. (we assume pull request is not reviewed properly) In the mining future data approach, these behaviors can be analyzed to detect and eliminate potentially invalid instances in the ground truth dataset. The elimination of problematic samples (i.e. non-ideal reviewer assignment) in the dataset will potentially increase the validity of the code reviewer selection process.

VI. Conclusion

In this study, we investigated the validity of ground truth for code reviewer recommendation studies. Most of the code reviewer recommendation studies that use real-life data assume that an assigned reviewer for a code changeset is the ideal reviewer. In Section IV.B, we demonstrated that the real-life reviewer assignments are not always done with the goal of finding the ideal reviewer. We further characterized these cases as *availability reasons* and *attribute substitution bias*. The discrepancy between real-life reviewer assignments and recommendation models raises a threat to the validity of the recommendation studies. The analysis of Ruangwan et al. [36] with four large software projects also supports our idea of the problematic ground truth in reviewer data related to the availability reasons. In the future, we would like to introduce quantitative evidence for attribute substitution bias as well.

To the best of our knowledge, the validity of the ground truth for the code reviewer recommendation studies has not been investigated in the literature. Our study examines this problem and explores possible solutions. The implications of this study are: (1) Previous reviewer recommendation studies should be reviewed in terms of the validity of the ground truth and success metrics. (2) New recommendation models and datasets should be created by considering this validity problem. As future work, we are planning to explore alternative solutions to this problem.

References

[1] M. Fagan, "Software Pioneers," *Softw. Pioneers*, pp. 214–225, 2011.

[2] M. L. de Lima Júnior, D. M. Soares, A. Plastino, and L. Murta, "Developers assignment for analyzing pull requests," *SAC '15 Proc. 30th Annu. ACM Symp. Appl. Comput. Pages*, pp. 1567–1572, 2015.

[3] M. L. de L. Júnior, D. M. Soares, A. Plastino, and L. Murta, "Automatic assignment of integrators to pull requests: The importance of selecting appropriate attributes," *J. Syst. Softw.*, vol. 144, pp. 181–196, 2018.

[4] K. H. Yang, T. L. Kuo, H. M. Lee, and J. M. Ho, "A reviewer recommendation system based on collaborative intelligence," *Proc. - 2009 IEEE/WIC/ACM Int. Conf. Web Intell. WI 2009*, vol. 1, pp. 564–567, 2009.

[5] J. Lipčák and B. Rossi, "A large-scale study on source code reviewer recommendation," *Proc. - 44th Euromicro Conf. Softw. Eng. Adv. Appl. SEAA 2018*, pp. 378–387, 2018.

[6] A. Ouni, R. G. Kula, and K. Inoue, "Search-based peer reviewers recommendation in modern code review," *Proc. - 2016 IEEE Int. Conf. Softw. Maint. Evol. ICSME 2016*, pp. 367–377, 2017.

[7] J. Jiang, J. H. He, and X. Y. Chen, "CoreDevRec: Automatic Core Member Recommendation for Contribution Evaluation," *J. Comput. Sci. Technol.*, vol. 30, no. 5, pp. 998–1016, 2015.

[8] G. Jeong, S. Kim, T. Zimmermann, and K. Yi, "Improving code review by predicting reviewers and acceptance of patches," *Rosaec Memo*, 2009.

[9] Z. Xia, H. Sun, J. Jiang, X. Wang, and X. Liu, "A hybrid approach to code reviewer recommendation with collaborative filtering," *SoftwareMining 2017 - Proc. 2017 6th IEEE/ACM Int. Work. Softw. Mining, co-located with ASE 2017*, pp. 24–31, 2017.

[10] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, p.2., 1997.

[11] C. Bird, V. Kovalenko, A. Bacchelli, N. Tintarev, and E. Pasynkov, "Does reviewer recommendation help developers?," *IEEE Trans. Softw. Eng.*, pp. 1–1, 2018.

[12] A. Bosu and J. C. Carver, "Impact of peer code review on peer impression formation: A survey," *Int. Symp. Empir. Softw. Eng. Meas.*, pp. 133–142, 2013.

[13] M. Beller, A. Bacchelli, A. Zaidman, and E. Juergens, "Modern code reviews in open-source projects: which problems do they fix?," *Proc. 11th Work. Conf. Min. Softw. Repos.*, pp. 202–211, 2014.

[14] M. E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Syst. J.*, vol. 15, no. 3, pp. 182–211, 2010.

[15] P. M. Johnson and D. Tjahjono, "Does Every Inspection Really Need a Meeting?," *Empir. Softw. Eng.*, vol. 3, no. 1, pp. 9–35, 1998.

[16] A. Bacchelli, E. Söderberg, M. Sipko, L. Church, and C. Sadowski, "Modern code review," *ICSE-SEIP '18 Proc. 40th Int. Conf. Softw. Eng. Softw. Eng. Pract.*, pp. 181–190, 2018.

[17] T. Baum, H. Lecmann, and K. Schneider, "The Choice of Code Review Process: A Survey on the State of the Practice," *Prod. Softw. Process Improv. PROFES 2017.*, vol. 10611 LNCS, pp. 111–127, 2017.

[18] X. Xia, D. Lo, X. Wang, and X. Yang, "Who should review this change?: Putting text and file location analyses together for more accurate recommendations," *2015 IEEE 31st Int. Conf. Softw. Maint. Evol. ICSME 2015 - Proc.*, pp. 261–270, 2015.

[19] P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida, and K. Matsumoto, "Who Should Review My Code ?," *Proc. 22nd IEEE Int. Conf. Softw. Anal. Evol. Reengineering*, pp. 141–150, 2015.

[20] C. Yang *et al.*, "RevRec: A two-layer reviewer recommendation algorithm in pull-based development model," *J. Cent. South Univ.*, vol. 25, no. 5, pp. 1129–1143, 2018.

[21] J. Jiang, Y. Yang, J. He, X. Blanc, and L. Zhang, "Who should comment on this pull request? Analyzing attributes for more accurate commenter recommendation in pull-based development," *Inf. Softw. Technol.*, vol. 84, pp. 48–62, 2017.

[22] M. Fejzer, P. Przymus, and K. Stencel, "Profile based recommendation of code reviewers," *J. Intell. Inf. Syst.*, vol. 50, no. 3, pp. 597–619, 2018.

[23] M. M. Rahman, C. K. Roy, J. Redl, and J. A. Collins, "CORRECT: code reviewer recommendation at GitHub for Vendasta technologies," *Proc. 31st IEEE/ACM Int. Conf. Autom. Softw. Eng.*, pp. 792–797, 2016.

[24] M. M. Rahman, C. K. Roy, and J. A. Collins, "CoRReCT: code reviewer recommendation in GitHub based on cross-project and technology experience," *2016 IEEE/ACM 38th Int. Conf. Softw. Eng. Companion*, pp. 222–231, 2016.

[25] V. Balachandran, "Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation," *Proc. - Int. Conf. Softw. Eng.*, pp. 931–940, 2013.

[26] M. B. Zanjani, H. Kagdi, and C. Bird, "Automatically Recommending Peer Reviewers in Modern Code Review," *IEEE Trans. Softw. Eng.*, vol. 42, no. 6, pp. 530–543, 2016.

[27] J. Kim and E. Lee, "Understanding review expertise of developers: A reviewer recommendation approach based on latent Dirichlet allocation," *Symmetry (Basel).*, vol. 10, no. 4, pp. 5–7, 2018.

[28] Y. Yu, H. Wang, G. Yin, and T. Wang, "Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment?," *Inf. Softw. Technol.*, vol. 74, pp. 204–218, 2016.

[29] Y. Yu, H. Wang, G. Yin, and C. X. Ling, "Reviewer recommender of pull-requests in GitHub," *Proc. - 30th Int. Conf. Softw. Maint. Evol. ICSME 2014*, pp. 609–612, 2014.

[30] Y. Yu, H. Wang, G. Yin, and C. X. Ling, "Who should review this pull-request: Reviewer recommendation to expedite crowd collaboration," *Proc. - Asia-Pacific Softw. Eng. Conf. APSEC*, vol. 1, pp. 335–342, 2014.

[31] J. B. Lee, A. Ihara, A. Monden, and K. I. Matsumoto, "Patch reviewer recommendation in OSS projects," *Proc. - Asia-Pacific Softw. Eng. Conf. APSEC*, vol. 2, pp. 1–6, 2013.

[32] E. Sulun, E. Tuzun, and U. Dogrusoz, "Reviewer Recommendation using Software Artifact Traceability Graphs," in *15th International Conference on Predictive Models and Data Analytics in Software Engineering*, 2019, in press.

[33] R. Mohanani, I. Salman, B. Turhan, P. Rodriguez, and P. Ralph, "Cognitive Biases in Software Engineering: A Systematic Mapping Study," *IEEE Trans. Softw. Eng.*, vol. PP, no. 99, 2018.

[34] C. Bird *et al.*, "Fair and Balanced ? Bias in Bug-Fix Datasets Categories and Subject Descriptors," *Proc. 7th Jt. Meet. Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng.*, pp. 121–130, 2009.

[35] M. Greiler, C. Bird, J. Czerwonka, L. MacLeod, and M.-A. Storey, "Code Reviewing in the Trenches: Challenges and Best Practices," *IEEE Softw.*, vol. 35, no. 4, pp. 34–42, 2017.

[36] S. Ruangwan, P. Thongtanunam, A. Ihara, and K. Matsumoto, "The impact of human factors on the participation decision of reviewers in modern code review," *Empir. Softw. Eng.*, pp. 1–43, 2018.

[37] O. Kononenko, O. Baysal, L. Guerrouj, Y. Cao, and M. W. Godfrey, "Investigating code review quality: Do people and participation matter?," *2015 IEEE 31st Int. Conf. Softw. Maint. Evol. ICSME 2015 - Proc.*, pp. 111–120, 2015.