MANAGEMENT OF CODE REVIEWER RECOMMENDATION USING OPTIMIZATION ALGORITHM, NSGA-III

Zaeem Anwaar and Wasi Haider Butt
Department of Computer and Software Engineering, College of Electrical and Mechanical Engineering (CEME), National University of Sciences and Technology (NUST)
Islamabad, Pakistan

ABSTRACT

Code review is considered as efficient and effective practice to improve software quality, identify and remove defects before integration. Code reviewers having right expertise, experience and apt amount of knowledge with the code being reviewed leads to successful code processes, fewer bugs and less maintenance cost. With the growing size of distributed development teams, picking suitable reviewers is a challenging task. However, due to less resources and shorter deadlines, the management of code reviews and appropriate recommendation of code reviewers based on three objectives consecutively is an ambitious task to be considered as aim of this research. This paper addresses the formulation for managing and recommending code reviewers based on multi conflicting objectives (i.e., availability, expertise and collaboration) simultaneously. Non-sorting genetic algorithm-III (NSGA-III) is used as optimization algorithm to find the most suitable reviewers while keeping expertise and availability ratio high and less collaboration between reviewers and developers. The results were implemented and validated on three (medium to large size) open-source projects named as LibreOffice, Qt and Open-Stack. We calculated precision, recall, mean reciprocal rank (MRR), accuracy for all 3 projects on average. The results from our proposed approach accurately recommended the code reviewers with the precision up to 80%, 86% of recall, 82% MRR and 84% accuracy by improving state-of-the-art. NSGA-III recommended the reviewers in less execution time and better fitness values in comparison to NSGA-II in all experimental sets. The proposed approach could be practical to Modern Code Review (MCR) in order to help developers while recommending suitable code-reviewers in less time and resources to speed up the review process.

KEYWORDS

Code-Reviewer Recommendation, Modern Code Review, Software Development, Multi-Objective Algorithm, NSGA-III

1. INTRODUCTION

Software code review is an integral part of software development and has been in practice for more than three decades. It involves identifying and fixing the defects i.e., logical errors or bugs in a software system, to ensure code quality. A reviewer is requested to review the change and identify the issues with the change, and then recommend further actions to the software developer responsible for the code (Asthana et al., 2019). A series of meetings between the reviewer and the developer are taken place, to ensure a mutual understanding of the change made and the review feedback. The drawback of this manual process of code review is time-consuming and expensive; as the time, effort and experience of the reviewers are not judiciously and efficiently utilized. The purpose is to get a code check-in the shortest possible time as the author/company needs to release the new version of the modified code as early as possible. Thus, a suitable code reviewer is required to serve the purpose of code review. Such a reviewer should possess thorough knowledge, experience and expertise needed for the job and shouldn't be over-committed as well. Only, the reviewer with the requisite expertise and appropriate time may contribute toward efficient examination of the code changes and defects (Chouchen, Ouni, Mkaouer, Kula, & Inoue, 2021).

Over the period, the manual process of assigning a reviewer has evolved into an informal, fully automated, structured and documented approach. It has progressed to a lightweight, quicker and tool-based process named Modern Code Review (Balachandran, 2013). MCR is also known as change-based code

review. MCR is a collaborative, quicker and automated approach that ensures that both author and code reviewer follow the standards of code review in a literal manner. Here, the reviewer is assigned to review a specific code, based on certain logic and certain credentials, in an automated manner. Some of the benefits of choosing the reviewer in an automated manner are: **1)** Reviewer is automatically assigned as per certain credentials in a much shorter time frame without compromising other projects. **2)** Human factor / biases which may result in a selection of inappropriate reviewers is eliminated. **3)** All the reviewers share the optimized load. **4)** The selection of the right reviewer (as-per requisite skills, experience and commitment) enhances the quality of the review (Chouchen et al., 2021), (Balachandran, 2013).

This paper proposes to articulate the selection of peer code reviewers as a multi-objective problem using an optimization algorithm 'NSGA-III' (Bhesdadiya, Trivedi, Jangir, Jangir, & Kumar, 2016), (Deb, 2014). Multi-objective optimization is defined as "to find the trade-off balanced optimization between more than one objective". Usually, it is difficult to find such a solution that provides multi-objective optimization because the objectives in the problem are mostly conflicting. To find and propose such solutions, mostly Genetic Algorithms are used. These algorithms consider certain credentials i.e., reviewer's profile, experience, workload, expertise and commitment status, etc. to recommend reviewers. These credentials are termed 'Objectives' and an algorithm may use a single objective, may incorporate multiple objectives, or may utilize many objectives to suggest a reviewer. For example, an algorithm considering only one objective (i.e., experience) may be termed a single objective utilization algorithm. The algorithm that takes into account two or three objectives is termed a multi-objective optimization algorithm and an algorithm taking into account more than three objectives, while deciding on a reviewer, is termed a many-objective algorithm.

The research motivation is to recommend/choose the right reviewers for the code review process more quickly and accurately to save time and resources. Moreover, we validated our approach on 3 open-source projects to confirm its efficiency and accuracy to comparison to the state of the art. The major objectives of the research are as follows: **1)** To reform and organize 3 different open-source project data-sets, that are to be used for Code reviewer recommendation (CRR). **2)** To perform a detailed literature review of the recent research on CRR. **3)** Explore Multi-objective optimization search algorithms. **4)** Propose an approach that navigates between three different proportions/credentials by providing multiple non-dominant peer reviewer recommendations instead of one solution. **5)** Analyzing and validating the precision, recall, MRR and Accuracy of the proposed approach. **6)** Comparison of the results with state-of-the-art.

The rest of the paper is followed as; Section II presents the literature of the related studies. Section III proposes the approach used in the research. Section IV discuss and presents the results. Lastly the paper is concluded.

2. LITERATURE REVIEW

To highlight some related researches, existing work or research that has been done concerning the proposed domain by worthy researchers is analyzed.

Authors (Chouchen et al., 2021) introduced multi-objective search-based approach 'Who-Review' to find the best suitable reviewers for code changing using IBEA (Indicator-Based Evolutionary Algorithm). The recommendation was based on two factors; experience and workload on reviewers. Validation on four open-source projects resulted in an average precision of 68% and recall of 77%.

Researchers (Thongtanunam et al., 2015) proposed 'Rev-Finder' to help developers in finding appropriate reviewers using a file-location based approach. Recommendation was based on finding similarity with the previously reviewed files. The approach was evaluated on 4 different data sets that resulted in accurately recommended 79% of reviews.

In (Balachandran, 2013), an automatic static analysis approach was introduced by authors named 'Review Bot'. It uses various static analysis tools as output to automate the process of checking coding standard violations. A user study was taken into account to validate the tool. The tool also recommended reviewers based on change history of the code. Results showed recommendation accuracy better than the method based on file change history.

'RevRec' (Ouni, Kula, & Inoue, 2016) a search-based approach to suggested reviewers and also to support in decision making for code change submitter was proposed by authors. Genetic algorithm was used

to perform the research aim based on their expertise and past collaborations. Authors evaluated the approach on 3 opensource projects and resulted in 59% of precision and 74% of recall.

The most recent and relevant literature to our approach is presented in (Rebai, Amich, Molaei, Kessentini, & Kazman, 2020). The authors formulated the recommendation of code reviewers as a multi-objective search problem to balance the conflicting objectives of expertise, availability, and history of collaborations. Authors used NSGA-II genetic algorithm in their study. 9 open-source projects were used to validate their study. The motivation for our approach is gathered from this study (Rebai et al., 2020) where we have updated the optimization algorithm by using NSGA-III. The authors from an open-source framework 'Pymoo' (Blank & Deb, 2020) for multi-objective optimization in python (Deb & Jain, 2013), (Jain & Deb, 2013) claim that NSGA-II is not much better in optimization of multi-objective (more than two objectives) as compare to NSGA-III (Blank, Deb, & Roy, 2019).

To deal with a large number of possible reviewers for multiple pull requests in terms of multi-objective context is a problem that is under-studied in the research literature. This management process requires handling multiple competing criteria including expertise, availability and previous collaborations with the owners and reviewers.

3. PROPOSED METHODOLOGY

In this section, the recommendation of a most appropriate set of reviewers is presented for pull requests to be reviewed as a framework/approach by using the NSGA-III optimization algorithm. The proposed approach consists of data collection, data pre-processing, main components of the approach, detailed knowledge about NSGA-III (i.e., high-level pseudo-code) algorithm, solution representation, fitness functions and change operators. The high-level diagram for proposed approach is shown in Figure 1. The methodology starts with a project's new pull request to be reviewed is received. The data is collected from the project in terms of its previous pull request information (newly opened, under-review, closed), reviewers' and developers' information, etc.

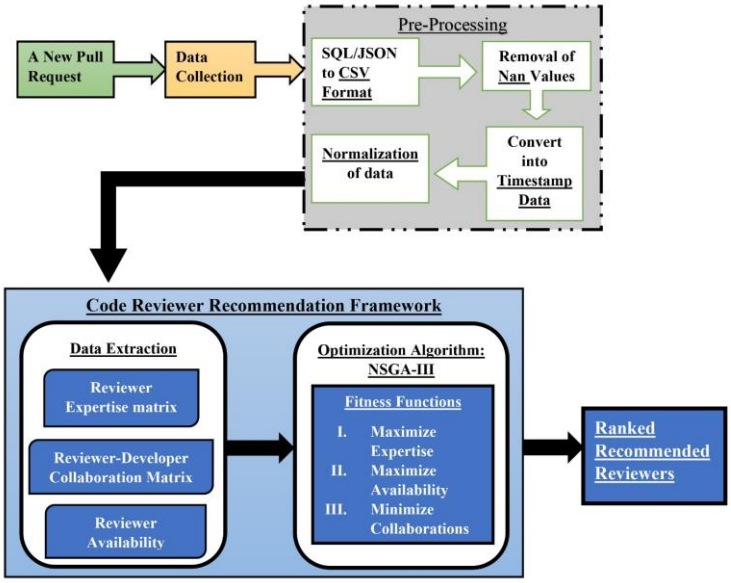

Figure 1. Proposed Approach

3.1 Data Pre-Processing

The data is collected from the link shared by the authors in the literature available at Mining Code Review Repositories (kin-y.github.io) in SQL format. None of the literature shared the data in CSV format. The gathered SQL data was converted into CSV format as it is faster to handle, a standard format and easy to

parse. Removal of Not-a-Number (NaN) values from the data was performed by replacing it with zero. The purpose of doing NaN removal was because optimization algorithm's performance and accuracy can't be made sure on NaN data. Also, the data was normalized by doing Min-Max scaling from 0 to 1 so that any outliers from the data-set should be removed. The pre-processed data in CSV format is shared at https://drive.google.com/file/d/1CiZWHc0z_JFZ_Tg7pewTfNZb91o9XzIg/view?usp=sharing as shown in Table 1.

Table 1. Project's Information

Project name	**Repository link**	**Project Duration**	**Num of pull request (Closed)**	**Num of pull request (New-Past 7 days)**	**Num of reviewers**
LibreOffice	https://git.libreoffice.org/core	07/2014~12/2021	28030	12	934
Qt	https://code.qt.io/cgit/qt/qtbase.git/	05/2011~11/2021	115888	25	1437
OpenStack	https://opendev.org/openstack	07/2011~12/2021	173749	52	5091

3.2 Code Reviewer Recommendation Framework

This section presents the framework in two steps. Step 1 explains the data extraction from the CSV and step 2 explains the optimization algorithm i.e., NSGA-III and fitness functions (Rebai et al., 2020).

3.2.1 Data Extraction

Three type of matrix formation is implemented to extract data that is required in our approach. Reviewer-Expertise Matrix, Reviewer-Developer Matrix and Reviewer Availability Matrix. Reviewer-Expertise Matrix represents the expertise of the reviewer based on previous commits that has been reviewed by reviewer. The number of times a reviewer is familiar/reviewed a same type of file is counted as his/her expertise. Reviewer-Developer is basically a collaboration matrix for the times when both collaborated on the same files i.e., closed pull requests. Lastly the reviewer availability matrix is formed according to the workload/files being reviewed on current time by the reviewer.

3.2.2 Fitness Functions

Since we are working on a multi-objective optimization so in our approach, we intend to provide optimization on three fitness functions. These three objective functions are Expertise, Availability, and Collaboration. Availability is defined as the inverse of approximated wait until reviewers from their workload (already working on a set of file S) become available. Expertise is estimated by the rank of a reviewer in the set of pool between other reviewers for reviewing a file. Lastly, Collaboration is the summation of all associates between the recommended reviewers chosen to work with a selected set of developers.

We aim to maximize the formulation of expertise and availability of the reviewers while minimizing the fitness function of collaboration due to socio-technical aspects of reviewers and developers in the hope to reduce human biases. The fitness functions are kept the same to literature studies so that a healthy comparison could be brought out of it. The formulas for calculating fitness functions can be explored at (Rebai et al., 2020). All these three-fitness function first values would be collectively passed to NSGA-III a best solution from all of them. The high-level pseudo-code of NSGA-III is available at (Bhesdadiya et al., 2016). The proposed approach is implemented in Python Language on Jupiter Lab (3.0) using Pymoo Library (Blank & Deb, 2020) in our case. While defining problem and Pymoo algorithm library, the change or genetic operators are used. We applied Das-Dennis (Bhesdadiya et al., 2016) approach to define reference points. The rest of the genetic operator used in results are explained in Table 2. The literature results are computed using random selection (to compare with state of the art), we have used tournament selection in one experiment (to give a new horizon for future comparison). We used 'Uniform ('real ux', 'bin ux', 'int ux') Crossover' and 'Polynomial ('real pm', 'int pm') Mutation' to explore and exploit the search space.

Table 2. Experimental Genetic Operators

Experiment Number	Pop Size	Selection Operator	Crossover Operator	Crossover Probability	Mutation Operator	Mutation rate	Number of Generation
Exp. 1	50	Random	Uniform	0.5	Polynomial	0.01	500
Exp. 2	50	Random	Uniform	0.6	Polynomial	0.1	450
Exp. 3	100	Random	Uniform	0.8	Polynomial	0.2	550
Exp. 4	200	Random	Uniform	0.5	Polynomial	0.1	550
Exp. 5	200	Random	Uniform	0.8	Polynomial	0.05	600
Exp. 6	100	Tournament	Uniform	0.85	Polynomial	0.2	500

4. RESULTS AND DISCUSSION

Precision, recall, Mean Reciprocal Rank (MRR) and average accuracy were calculated to compare our results with state-of-the-art.

$$Pricision@Exp = \frac{TP}{TP + FP} \quad (1) \qquad Recall@Exp = \frac{TP}{TP + FN} \quad (2)$$

In equation (1) and (2), TP (True Positive) corresponds to the number of top-k reviewers recommended by the approach and also actual reviewers. FP (False Positive) corresponds to the number of top-k reviewers recommended by the approach, but not actual reviewers; FN (False Negative) corresponds to the number of actual reviewers, that are not among the top-k reviewers recommended by the approach.TN (True Negative) corresponds to the number of not actual reviewers, that are also not among the top-k reviewers recommended by the approach.

MRR (Mean Reciprocal Rank): The average rank of correct reviewers in the recommendation list. Given a reviewers recommendation lists R, the score MRR is calculated with the help of equation (3).

$$MRR = \frac{|\Sigma \forall r \epsilon R| \text{rank}(r)}{|R|} \quad (3)$$

where rank(r) is the rank score of the first reviewer in the recommendation list r. The higher is the MRR score, the better is the recommendation rank approach.

Average Accuracy of Projects: The projects accuracy is determined as average accuracy in terms of number of experiments performed in that project. The average accuracy Equation. (4) is determined as:

$$Average\ Accuracy = \frac{\Sigma \text{Accuracy}(E1 + E2 \ldots EN)}{N} \quad (4)$$

Where E1 is the accuracy of experiment 1, E2 is the accuracy of experiment 2 till last Experiment. The N is the total number of Experiments used in a project.

The efficiency and precision to identify relevant code reviewers by using of our proposed approach i.e., NSGA-III is confirmed on pull requests from 3 different projects are resulted in table III IV and figure no 2 & 3. Tables 3 and 4 shows the precision and recall in context to every experiment result separately. For example, LibreOffice has a precision ranging from 51% to 67% for all experiments. It has a recall range from 52% to 75%. Qt project has precision range of 50% to 69% and recall rate ranging from 58% to 73%. Whereas Open-Stack project has the highest precision value of 80% and recall ranging from 69-86%. Due to large number of reviewers in the projects (i.e., Qt, LibreOffice) the precision rate of 50% or 51% in projects could also be considered acceptable. Also, some of the highest recall scores are obtained in Open-Stack as some pull request require more than one code reviewer. The highlighted/bold results in both the tables are the best ones identified.

Table 3. Precision Comparison with literature

Project Name	Experiment number	Precision@Exp.					
		Proposed Approach (NSGA-III)	**AEC (NSGA-II)**	**RevRec (GA)**	**Who Review (IBEA)**	**RevFinder**	**ReviewBot**
LibreOffice	**Exp. 1**	**0.64**	N/A	0.52	0.61	0.48	0.38
	Exp. 2	**0.59**	N/A	0.45	0.54	0.4	0.36
	Exp. 3	**0.57**	N/A	0.50	0.56	0.42	0.40
	Exp. 4	0.52	N/A	0.41	**0.53**	0.32	0.32
	Exp. 5	**0.51**	N/A	0.39	0.46	0.3	0.23
	Exp. 6	**0.67**	N/A	N/A	N/A	N/A	N/A
Qt	**Exp. 1**	**0.66**	0.58	0.49	0.58	0.3	0.22
	Exp. 2	**0.62**	0.51	0.42	0.53	0.27	0.19
	Exp. 3	**0.57**	0.54	0.45	0.55	0.29	0.13
	Exp. 4	**0.55**	0.52	0.41	0.43	0.21	0.10
	Exp. 5	**0.50**	0.46	0.34	0.48	0.16	0.09
	Exp. 6	**0.69**	N/A	N/A	N/A	N/A	N/A
OpenStack	**Exp. 1**	**0.74**	0.70	0.59	0.62	0.32	0.24
	Exp. 2	**0.69**	0.64	0.57	0.55	0.27	0.2
	Exp. 3	**0.67**	0.65	0.51	0.59	0.30	0.22
	Exp. 4	**0.63**	**0.63**	0.43	0.54	0.25	0.16
	Exp. 5	**0.57**	0.54	0.46	0.48	0.21	0.11
	Exp. 6	**0.80**	N/A	N/A	N/A	N/A	N/A

Table 4. Recall Comparison with literature

Project Name	Experiment number	Recall@Exp.					
		Proposed Approach (NSGA-III)	**AEC (NSGA-II)**	**RevRec (GA)**	**Who Review (IBEA)**	**RevFinder**	**ReviewBot**
LibreOffice	**Exp. 1**	**0.67**	N/A	0.34	0.48	0.32	0.18
	Exp. 2	**0.62**	N/A	0.48	0.52	0.38	0.22
	Exp. 3	**0.59**	N/A	0.57	0.56	0.42	0.20
	Exp. 4	0.59	N/A	0.58	**0.61**	0.45	0.31
	Exp. 5	**0.69**	N/A	0.59	0.68	0.49	0.38
	Exp. 6	**0.75**	N/A	N/A	N/A	N/A	N/A
Qt	**Exp. 1**	**0.58**	0.56	0.41	0.44	0.14	0.09
	Exp. 2	**0.62**	0.60	0.50	0.45	0.27	0.16
	Exp. 3	**0.68**	0.66	0.55	0.58	0.30	0.20

	Exp. 4	**0.72**	0.68	0.59	0.60	0.35	0.24
	Exp. 5	**0.73**	0.70	0.65	0.64	0.43	0.30
	Exp. 6	**0.67**	N/A	N/A	N/A	N/A	N/A
OpenStack	**Exp. 1**	**0.69**	0.59	0.31	0.46	0.15	0.12
	Exp. 2	**0.70**	0.68	0.39	0.49	0.29	0.2
	Exp. 3	**0.78**	0.76	0.52	0.59	0.37	0.32
	Exp. 4	**0.80**	0.75	0.54	0.60	0.46	0.39
	Exp. 5	**0.82**	0.80	0.66	0.63	0.50	0.41
	Exp. 6	**0.86**	N/A	N/A	N/A	N/A	N/A

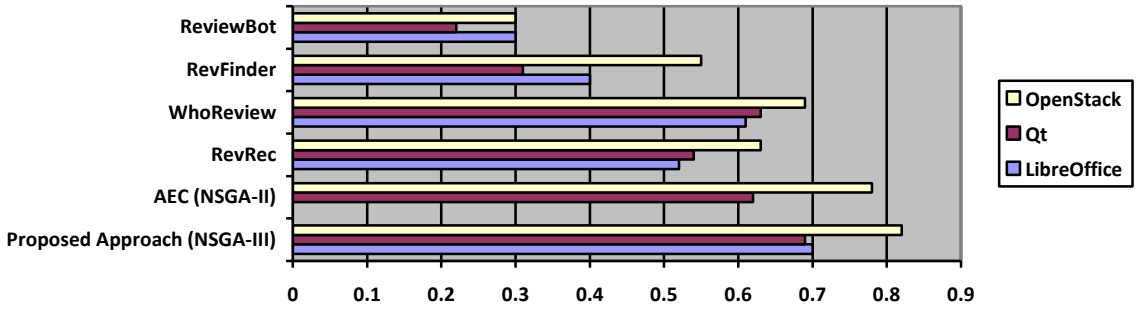

Figure 2. Mean Reciprocal Rank Chart

Figure 2 shows the MRR values that NSGA-III was able to efficiently rate the recommended code reviewers. The best resulted solution of the last population obtained in the last iteration of GA are copied in a single pool. Then, the rank of each reviewer corresponds to his frequency count in the pool. That is, reviewers that are recommended in many solutions are ranked first. The proposed approach shows the MRR values better than all other presented in literature. LibreOffice came up with 70%, Qt with 69% and Open-Stack with highest 82% MRR scores. The efficiency for ranking the reviewers is measured by this parameter so the outcomes of MRR are very important. Additionally, it is one of the main motivations of this proposed approach as ranking is directly related to availability and collaboration of code reviewers.

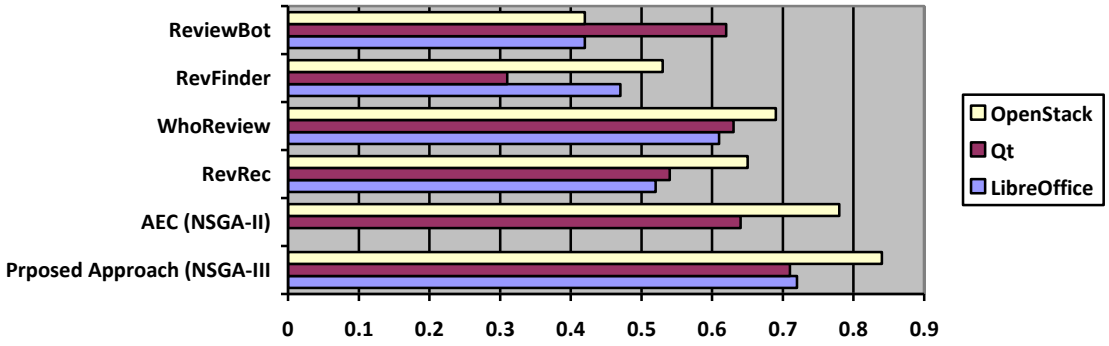

Figure 3. Average Accuracy Chart

Figure 3 presents the average accuracy of our approach on experiments. The accuracy of each experiment is calculated separately and then average of accuracy's is calculated as discussed in equations 9 and 10. LibreOffice came up with 72%, Qt with 71% and Open-Stack with highest 84% Average Accuracy scores which are better than all others approach.

Most importantly our proposed technique doesn't have a bias towards the projects that are used for validation as we used average values of mean reciprocal rank and accuracy. Out of 3 open-source projects, our proposed approach performed well on Open-Stack pull requests. The accuracy, MRR, precision and recall of the project was good in comparison to other projects.

5. CONCLUSION

In this research, we have proposed a multi objectives problem to manage and recommend code reviewers by adopting an optimization algorithm that is NSGA-III. The purpose is to recommend the best trade-off reviewers between three conflicting objectives i.e., maximizing the availability and expertise of reviewers and minimizing the collaboration between developers and reviewers to lessen the human biases factor. We implemented and evaluated our approach on three (medium to large size) open-source projects named as LibreOffice, Qt and Open-Stack. We calculated efficiency on our approach by finding precision, recall, MRR, accuracy for all 3 projects on average. The results from our proposed approach accurately recommended the code reviewers with the precision up to 80%, 86% of recall, 82% mean reciprocal rank and 84% average accuracy by improving state-of-the-art. The proposed approach could be practical to MCR in order to help developers while recommending suitable code-reviewers in less time and resources to speed up the review process. This research highlighted the importance of managing code reviews to reduce delays in review process in less time and resources while confirming high expertise and availability as much as possible.

REFERENCES

Asthana, S., Kumar, R., Bhagwan, R., Bird, C., Bansal, C., Maddila, C., ... Ashok, B. (2019). *WhoDo: automating reviewer suggestions at scale.* Paper presented at the Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.

Balachandran, V. (2013). *Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation.* Paper presented at the 2013 35th International Conference on Software Engineering (ICSE).

Bhesdadiya, R. H., Trivedi, I. N., Jangir, P., Jangir, N., & Kumar, A. J. C. E. (2016). An NSGA-III algorithm for solving multi-objective economic/environmental dispatch problem. *3*(1), 1269383.

Blank, J., Deb, K., & Roy, P. C. (2019). *Investigating the normalization procedure of NSGA-III.* Paper presented at the International Conference on Evolutionary Multi-Criterion Optimization.

Blank, J., & Deb, K. J. I. A. (2020). Pymoo: Multi-objective optimization in python. *8*, 89497-89509.

Chouchen, M., Ouni, A., Mkaouer, M. W., Kula, R. G., & Inoue, K. J. A. S. C. (2021). WhoReview: A multi-objective search-based approach for code reviewers recommendation in modern code review. *100*, 106908.

Deb, K. (2014). Multi-objective optimization. In *Search methodologies* (pp. 403-449): Springer.

Deb, K., & Jain, H. J. I. t. o. e. c. (2013). An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. *18*(4), 577-601.

Jain, H., & Deb, K. J. I. T. o. e. c. (2013). An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part II: Handling constraints and extending to an adaptive approach. *18*(4), 602-622.

Ouni, A., Kula, R. G., & Inoue, K. (2016). *Search-based peer reviewers recommendation in modern code review.* Paper presented at the 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME).

Rebai, S., Amich, A., Molaei, S., Kessentini, M., & Kazman, R. J. A. S. E. (2020). Multi-objective code reviewer recommendations: balancing expertise, availability and collaborations. *27*(3), 301-328.

Thongtanunam, P., Tantithamthavorn, C., Kula, R. G., Yoshida, N., Iida, H., & Matsumoto, K.-i. (2015). *Who should review my code? a file location-based code-reviewer recommendation approach for modern code review.* Paper presented at the 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER).