Recommending Peer Reviewers in Modern Code Review : A Multi-Objective Search-based Approach

Motaz Chouchen
ETS Montreal, University of Quebec
Montreal, QC, Canada
moataz.chouchen.1@ens.etsmtl.ca

Ali Ouni
ETS Montreal, University of Quebec
Montreal, QC, Canada
ali.ouni@etsmtl.ca

Mohamed Wiem Mkaouer
Rochester Institute of Technology
Rochester, NY, USA
mwmvse@rit.edu

Raula Gaikovina Kula
Nara Institute of Science and Technology, Nara, Japan
raula-k@is.naist.jp

Katsuro Inoue
Osaka University
Osaka, Japan
inoue@ist.osaka-u.ac.jp

ABSTRACT

Modern code review is a common practice used by software developers to ensure high software quality in open source and industrial projects. During code review, developers submit their code changes which should be reviewed, via tool-based code review platforms, before being integrated into the codebase. Then, reviewers provide their feedback to developers, and may request further modifications before finally accepting or rejecting the submitted code changes. However, the identification of appropriate reviewers is still a tedious task as the number of code reviews to be performed is inflated with the increasing number of code changes and the increasing size of software development teams in today's large and active software projects. To help developers with the review process, we introduce a multi-objective search-based approach to find the appropriate set of reviewers. We use the Non-dominated Sorting Genetic Algorithm (NSGA-II) to optimize two conflicting objectives (*i*) maximize reviewers expertise with the changed files, and (*ii*) minimize reviewers workload in terms of their current open code reviews. We conduct a preliminary evaluation on two open source projects to evaluate our approach. Results indicate that our approach is efficient as compared to state-of-the-art approaches.

CCS CONCEPTS

• **Software and its engineering** → Software maintenance tools; • **Human-centered computing** → Empirical studies in collaborative and social computing;

1 INTRODUCTION

Peer code review is a well-established and widespread software engineering practice that aims at ensuring software quality, and sustaining software development teams in both industrial and open-sourcesoftware (OSS) projects [1]. Modern Code Review (MCR) originates from the classic code inspection [4] where formal and in-person meetings are required to discuss about potential problem with the code changes before they are merged with the code repository. MCR fosters less formal, lightweight and tool-based code review where developers can submit, discuss and rework code changes prior merging them.

Finding appropriate code reviewers is a non-trivial decision-making task in software engineering involving several considerations. As modern software projects code-base and development teams size are in constant increase, reviewers identification is becoming more challenging. In typical software projects, source files could be edited by several contributors, and reviewed by several reviewers. Contributors and reviewers could perform different software engineering tasks with different workloads. It is thus more difficult to identify suitable peer reviewers especially when the number of changed modules/files is large and reviewers are overloaded with different submitted code changes. Various approaches have been proposed to recommend code reviewers relying mostly on the expertise with the code files being submitted for review [5, 6, 8]. However, most of these approaches deal with the peer reviewers recommendation problem from a single perspective, *i.e.*, reviewers expertise and/or past collaborations, ignoring an important aspect which is the *reviewers workload*. Hence, reviewers are recommended regardless of their number of outstanding reviews.

To address this issue, we extend our previous work [5] by introducing a new reviewers recommendation formulation as a multi-objective search based problem. Our approach aims at finding the set of suitable developers that provide an optimal trade-off between two objectives (1) reviewers expertise with the changed code files and (2) reviewers workload through balancing code review tasks among available reviewers. We use the Non-dominated Sorting Genetic Algorithm (NSGA-II) [3]. We present an evaluation of our approach on two OSS projects. Results indicate that our approach is efficient as compared to state-of-the-art approaches.

2 APPROACH

Our approaches takes as input (1) a submitted code change for review which consists of a set of changed files submitted by a given developer from the project's team, and (2) the history of code reviews collected from the project's review platform, *e.g.*, Gerrit. As output, our approach returns a set of recommended reviewers that are most appropriate to review the submitted code change. To search for the best set of candidate reviewers, our approach

GECCO '20 Companion, July 8–12, 2020, Cancún, Mexico
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-7127-8/20/07.
https://doi.org/10.1145/3377929.3390057

uses the Non-dominated Sorting Genetic Algorithm (NSGA-II) [3]. The search process aims at finding the best trade-off between two objectives :

- *Maximize the reviewers expertise and collaboration* : We use the reviewers expertise and collaboration model defined in [5]. It is calculated by leveraging (*i*) the number of comments in prior reviews on similar files based on the file path similarity while considering the recency factor, and (*ii*) the collaboration social network between the developer who submitted the code change and the contributors in the project.
- *Minimize the reviewers workload* : We calculate the workload of a reviewer based on the number of open reviews she/he is currently involved in (with at least one review comment). For each current code change, we calculate her/his workload in terms of (*i*) the number of changed files and (*ii*) the average time (in days) spent in reviewing prior code changes involving these files as an estimation of the required workload for reviewing the current file.

We use a vector encoding for the solution representation where each dimension corresponds to a random reviewer ID that is picked from the list of available reviewers in the project. For the change operators, we use the standard single-point cut crossover and a creep mutation where one or more random dimensions are selected and changed with random values (*i.e.*, reviewers).

3 EVALUATION

Experiment setup. We evaluate our approach on two OSS projects from an an existing benchmark[1] that were studied in recent MCR research literature [5–8], namely Android [2] and Qt[3]. Table 1 shows the dataset statistics. We compare the efficiency of approach with recent state-of-the-art techniques, RevRec [5], cHRev [8], and ReviewBot [2]. RevRec [5] uses a mono-objective search based approach to identify reviewers that more most experienced and have previous collaborations. cHRev [8] uses reviewers expertise model the history of reviews based on the review comments frequency and recency. RevFinder. ReviewBot [2] uses static analysis to build an expertise model based on the source code change history. Our comparison is based on the precision@k, recall@k and mean reciprocal rank (MRR) performance metrics used in prior works [5–8].

Table 1: Statistics of both studied systems Android and Qt.

System	#Reviews	#Reviewers	#Files	Av. files per review
Android	5,126	94	26,840	8.26
Qt	23,810	202	78,401	10.64

Results. Table 2 reports the average precision and recall results achieved by each of the studied approaches, NSGA-II, RevRec, cHRev and ReviewBot. We observe that NSGA-II achieves the highest precision among RevRec, cHRev and ReviewBot in both systems. In particular, for the top-1, it achieves the highest precision score at 0.67. We notice consistent results across both projects with a precision@1 of 0.67 and 0.57 for Android and Qt, respectively. Likewise, in terms of recall, our approach achieves a higher performance in the top-10 recommended reviewers with a score of 0.73 and 0.69 for Android and Qt, respectively. Moreover, Table 3 reports the MRR results reflecting the overall ranking performance. Our approach achieves superior MRR scores of 0.72 for Android and 0.61 for Qt as compared to RevRec, cHRev and ReviewBot. This indicated that our approach provides a higher chance of recommending appropriate reviewers in the first ranks. To get a more qualitative sense on the studied projects, we observe a high divergence in terms of the projects team sizes and the total number of reviewers available, *e.g.*, Qt has 202 reviewers, while Android has only 94. This finding indicates the good scalability of our approach with respect to the total number of available reviewers in a project.

[1] https://kin-y.github.io/miningReviewRepo
[2] https://source.android.com/
[3] http://qt-project.org/

Table 2: The average precision@k and recall@k results.

Project	**k**	**Precision@k**				**Recall@k**			
		NSGA-II	RevRec	cHRev	ReviewBot	NSGA-II	RevRec	cHRev	ReviewBot
Android	1	**0.67**	0.58	0.5	0.21	0.44	0.38	0.27	0.11
	3	0.61	0.47	0.35	0.17	0.52	0.51	0.5	0.19
	5	0.51	0.39	0.3	0.12	0.64	0.61	0.61	0.29
	10	0.47	0.34	0.26	0.09	**0.73**	0.71	0.65	0.38
Qt	1	**0.57**	0.49	0.45	0.22	0.45	0.41	0.33	0.09
	3	0.54	0.45	0.4	0.19	0.57	0.5	0.47	0.16
	5	0.49	0.41	0.37	0.13	0.61	0.59	0.52	0.24
	10	0.42	0.34	0.34	0.09	0.69	0.65	0.6	0.3

Table 3: The achieved MRR scores by each approach.

	NSGA-II	RevRec	cHRev	ReviewBot
Android	**0.72**	0.69	0.60	0.25
Qt	**0.61**	0.54	0.31	0.22

4 CONCLUSION AND FUTURE WORK

We introduced in this paper a multi-objective approach to find and recommend code reviewers for MCR using NSGA-II. Our approach aims at optimizing (1) the reviewers expertise and collaboration and (2) the reviewers workload. As a preliminary evaluation on two large OSS projects, we found that our approach outperforms recent approaches based ont the reviewers expertise in terms of precision, recall and MRR. As future work, we plan to evaluate our approach on a larger set of industrial and OSS projects. We plan also to extend the expertise, collaboration and workload models to consider additional socio-technical factors and extend our approach with an interactive component to consider developers preferences.

REFERENCES

[1] Alberto Bacchelli and Christian Bird. 2013. Expectations, Outcomes, and Challenges of Modern Code Review. In *35th International Conference on Software Engineering (ICSE)*. Piscataway, NJ, USA, 712–721.

[2] Vipin Balachandran. 2013. Reducing Human Effort and Improving Quality in Peer Code Reviews Using Automatic Static Analysis and Reviewer Recommendation. In *35th International Conference on Software Engineering (ICSE)*. 931–940.

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

[4] M. E. Fagan. 1976. Design and code inspections to reduce errors in program development. *IBM Systems Journal* 15, 3 (1976), 182–211.

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

[6] Thongtanunam Patanamon, T. Chakkrit, Gaikovina Kula Raula, Yoshida Norihiro, Iida Hajimu, and Matsumoto Ken-ichi. 2015. Who Should Review My Code? A File Location-Based Code-Reviewer Recommendation Approach for Modern Code Review. In *Int. Conf. on Software Analysis, Evolution, and Reengineering*.

[7] Xin Yang, Raula Gaikovina Kula, Norihiro Yoshida, and Hajimu Iida. 2016. Mining the Modern Code Review Repositories: A Dataset of People, Process and Product. In *13th Working Conference on Mining Software Repositories (MSR)*.

[8] Motahareh Bahrami Zanjani, Huzefa Kagdi, and Christian Bird. 2015. Automatically recommending peer reviewers in modern code review. *IEEE Transactions on Software Engineering* 42, 6 (2015), 530–543.