Suggesting Reviewers of Software Artifacts using Traceability Graphs

Emre Sülün
emre.sulun@ug.bilkent.edu.tr
Bilkent University
Ankara, Turkey

ABSTRACT

During the lifecycle of a software project, software artifacts constantly change. A change should be peer-reviewed to ensure the software quality. To maximize the benefit of review, the reviewer(s) should be chosen appropriately. However, choosing the right reviewer(s) might not be trivial especially in large projects. Researchers developed different methods to recommend reviewers. In this study, we introduce a novel approach for reviewer recommendation problem. Our approach utilizes the traceability graph of a software project and assigns a know-about score to each developer, then recommends the developers who have the maximum know-about score for an artifact. We tested our approach on an open source project and achieved top-3 recall of 0.85 with an MRR (mean reciprocal ranking) of 0.73.

CCS CONCEPTS

• **Software and its engineering → Software maintenance tools**; **Collaboration in software development**.

KEYWORDS

reviewer recommendation, code review, software traceability

ACM Reference Format:
Emre Sülün. 2019. Suggesting Reviewers of Software Artifacts using Traceability Graphs. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19), August 26–30, 2019, Tallinn, Estonia.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3338906.3342507

1 PROBLEM STATEMENT AND MOTIVATION

Change is a common property of software development. Throughout the development lifecycle; requirements, test cases, design diagrams and other artifacts may change frequently. For better software quality, a change should be reviewed and the precondition of review is assigning reviewer(s).

Among the other artifacts, source code files are the most changing ones. In the literature, state of the art in code review [5][3] and code reviewer recommendation approaches are heavily researched. Thongtanunam et al. finds that when code-reviewers are assigned manually, approval process takes 12 more days compared to automatic assignments [6]. In addition to reducing code review time, assigning the review to a proper developer also increases the review quality and reduces the potential errors related to the reviewed artifact. In summary, finding the appropriate reviewer automatically is an important problem in the industry. Researchers proposed different code reviewer recommendation algorithms using various methods such as Bayesian networks [1], genetic algorithms [4], text mining [7] and support vector machines [2]. To the best of our knowledge, there is no prior work on artifact reviewer suggestion using traceability graphs.

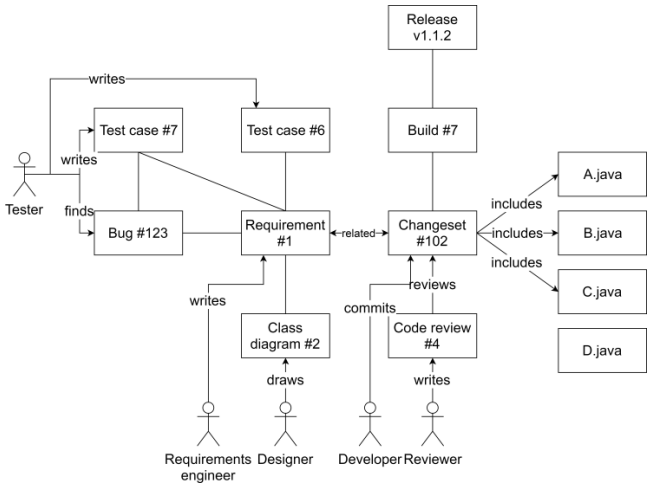

Figure 1: A typical artifact traceability graph for a software project

In this study, we introduce a novel approach for reviewer recommendation problem. The core of our approach relies on traceability graphs and relations between software artifacts. To illustrate our approach, we modeled a typical software development scenario that is shown in Figure 1. Example software artifacts, team members and relations between the software artifacts and team members are represented in the figure.

In this scenario, software artifacts may include requirements, design diagrams, use case diagrams, changesets, source code files, code reviews, test cases, bugs, builds or releases. The relations could be tests, includes, reviews, commits, etc. Team members, on the other hand, may include requirements engineers, designers, developers, reviewers or testers. Using traceability information of the project, a graph can be constructed. Artifacts and team members are represented as nodes and relations between them are represented

ESEC/FSE '19, August 26–30, 2019, Tallinn, Estonia

ACM ISBN 978-1-4503-5572-8/19/08.
https://doi.org/10.1145/3338906.3342507

as edges in such a graph. In the following section, we will describe how we utilized traceability graphs to recommend a reviewer.

2 APPROACH

We created an algorithm running on traceability graph of the software project. The algorithm basically assigns a score to a developer for each artifact. We called that score as know-about score. It measures the knowledge of a developer about an artifact. The higher know-about score means a higher knowledge about an artifact and a developer who has the highest know-about score should review the artifact.

For a given traceability graph, know-about score of a developer for an artifact is calculated according to the following formula.

$$KA_{Developer-Artifact} = \sum_{i} \frac{1}{Length(Path_i(Developer - Artifact))}$$

For example, if there are two paths from developer D to artifact A and if path lengths are 2 and 3, then know-about score of D for A is 1/2 + 1/3 = 0.83. We created this formula by claiming that the knowledge of a developer about an artifact is proportional to the number of paths and inversely proportional to the path length. Since reviews are usually done on a changeset instead of a single artifact change, we further generalize the formula for a set of artifacts. To find the know-about score of a developer for a changeset, we sum scores for each artifact in the changeset.

$$KA_{Developer-Changeset} = \sum_{i} KA_{Developer-Artifact(i)}$$

$$where\ Artifact(i) \in Changeset$$

To implement the graph, we used, Neo4j, an open source graph database. Neo4j allows us to store artifacts and relations, visualize them and run our algorithm with the help of Cypher query language.

3 CASE STUDY

3.1 About the Dataset

We tested our approach on an open source software project: Qt 3D Studio. We choose Qt 3D Studio because the associated data sets; Git history, issue tracking history (Jira) and code review history (Gerrit) are public. We fetched the history and created a traceability graph accordingly.

In Qt 3D Studio project, there are a total of 36 developers, 628 issues, 4151 files (including source code and binary files) and 942 commits at the time of fetching. Starting from the first commit, we expanded the graph commit by commit. For each commit, we run the algorithm and created a recommended reviewers list. Then, we compared actual reviewers list with the recommended reviewers list and calculated success metrics finally.

3.2 Results

We calculated top-k and mean reciprocal rank (MRR) metrics that are also commonly used by other reviewer recommendation algorithms [7][6][2]. In top-k metric, if one of the first k developers in the recommended reviewers list also appears in the actual reviewers list, it is counted as a successful recommendation. We calculated success rates for k = 1, 3 and 5. MRR is a value describing an average position of actual code reviewer in the list returned by recommendation algorithm. The results are summarized at Table 1.

Table 1: Top-k and MRR Metrics for QT 3D Studio Project

	Top-1	Top-3	Top-5	MRR
Score	0.63	0.85	0.86	0.73

Table 2: Comparison with Other Approaches

	Top-1	Top-3	Top-5
Our approach	0.63	0.85	0.86
TIE	0.52	0.73	0.79
RevFinder	0.32	0.55	0.64
CoreDevRec	0.57	0.82	0.88

We compared our results with three different approaches: TIE [7], RevFinder [6] and CoreDevRec [2]. Although metrics are the same, datasets that are used by other approaches are different. We could not use their datasets because they were not public. The comparison results are given at Table 2.

It can be stated that our approach produces promising results for recommending reviewers in terms of top-k metrics. We think the main reason is that other approaches usually do not consider other data sources such as issues history while we utilize the traceability graph that contains source code, review and issue data sources.

3.3 Limitations

While evaluating our approach, we used top-k and MRR metrics since they are commonly used by researchers. However, these metrics simply assume that the actual reviewers are the most suitable developers for an artifact, but this may not be valid for all review scenarios. Therefore, developing a more comprehensive metric may potentially help to remove this limitation.

The success of our approach depends on the availability of traceability data. Therefore, our approach may not work well in the early phases of a project where limited data related to traceability graph is present.

4 CONTRIBUTIONS AND FUTURE WORK

In this study, we developed a novel approach to suggest an appropriate reviewer for a changing artifact. The approach utilizes the traceability graph. We tested our approach on Qt 3D Studio, an open source project, and calculated top-k and MRR metrics. We summarized the main contributions of our study listed below;

- Our approach shows promising top-k and MRR scores compared to other methods. In the future, we are planning to test our approach on the datasets that other code reviewer recommendation approaches work.
- Our approach is not limited for only source code artifacts and can potentially be used for other types of artifacts such as recommending developers for fixing bugs.

REFERENCES

[1] Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. 2009. Improving code review by predicting reviewers and acceptance of patches. *Research on Software* (2009). http://rosaec.snu.ac.kr/publish/2009/techmemo/ROSAEC-2009-006.pdf

[2] Jing Jiang, Jia Huan He, and Xue Yuan Chen. 2015. CoreDevRec: Automatic Core Member Recommendation for Contribution Evaluation. *Journal of Computer Science and Technology* 30, 5 (2015), 998–1016. https://doi.org/10.1007/s11390-015-1577-3

[3] Laura MacLeod, Michaela Greiler, Margaret Anne Storey, Christian Bird, and Jacek Czerwonka. 2018. Code Reviewing in the Trenches: Challenges and Best Practices. *IEEE Software* (2018). https://doi.org/10.1109/MS.2017.265100500

[4] Ali Ouni, Raula Gaikovina Kula, and Katsuro Inoue. 2017. Search-based peer reviewers recommendation in modern code review. *Proceedings - 2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016* (2017), 367–377. https://doi.org/10.1109/ICSME.2016.65

[5] Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern code review. *Proceedings of the 40th International Conference on Software Engineering Software Engineering in Practice - ICSE-SEIP '18* (2018), 181–190. https://doi.org/10.1145/3183519.3183525

[6] Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Raula Gaikovina Kula, Norihiro Yoshida, Hajimu Iida, and Ken Ichi Matsumoto. 2015. Who should review my code? A file location-based code-reviewer recommendation approach for Modern Code Review. *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015 - Proceedings* (2015), 141–150. https://doi.org/10.1109/SANER.2015.7081824

[7] Xin Xia, David Lo, Xinyu Wang, and Xiaohu Yang. 2015. Who should review this change?: Putting text and file location analyses together for more accurate recommendations. *2015 IEEE 31st International Conference on Software Maintenance and Evolution, ICSME 2015 - Proceedings* (2015), 261–270. https://doi.org/10.1109/ICSM.2015.7332472