An Empirical Study of Reviewer Recommendation in Pull-based Development Model

Cheng Yang, Xunhui Zhang, Lingbin Zeng, Qiang Fan, Gang Yin and Huaimin Wang
National Laboratory for Parallel and Distributed Processing, College of Computer
Changsha, Hunan, China, 410073
delpiero710@126.com, zenglingbin12@163.com, {zhangxunhui,fanqiang09,yingang,hmwang}@nudt.edu.cn

ABSTRACT

Code review is an important process to reduce code defects and improve software quality. However, in social coding communities using the pull-based model, everyone can submit code changes, which increases the required code review efforts. Therefore, there is a great need of knowing the process of code review and analyzing the pre-existing reviewer recommendation algorithms. In this paper, we do an empirical study about the PRs and their reviewers in Rails project. Moreover, we reproduce a popular and effective IR-based code reviewer recommendation algorithm and validate it on our dataset which contains 16,049 PRs. We find that the inactive reviewers are very important to code reviewing process, however, the pre-existing method's recommendation result strongly depends on the activeness of reviewers.

KEYWORDS

pull request, code reviewer recommendation, GitHub

1 INTRODUCTION

Over 30 years, code review has been regarded as the best practice of software engineering both in industry and open source communities [2], which can help to improve the quality of source code. Traditionally, code reviewers are third-party developers who can identify the defects in source code before integrating into the system [16], and the code review process should be held in group meetings, which will cost a lot of time. With the development of social coding communities like GitHub, traditional code review is gradually replaced by modern code review [9]. When a code change is submitted to developers for reviewing, the reviewers will cooperate with each other to discuss and give suggestions about the code. If the change meets the need of the project and is supported by many reviewers, it will be integrated into the project's source code. However, code review actually costs a lot of time [15], and the reviewers invited are not always suitable for the job. Therefore, in order to reduce the time cost and improve the effectiveness of code review, it is necessary to recommend suitable reviewers to different code changes.

With the development of distributed software development, the pull-based model becomes more and more popular because it can lower the barrier of developers, which means that everyone can submit pull requests to the repository. Because the amount of code changes increases, open source projects need more developers to review the PRs and guard the quality of the project. As there are so many developers in open source repositories and the number increases sharply, it becomes more and more important to recommend suitable and qualified reviewers to PRs.

Looking at the above factors, we find that for modern code review especially pull-based models, it is necessary to recommend developers to review code changes. There have already existed many studies focusing on code reviewer recommendation. Jeong, Balachandran, Thongtanunam, Yu et al.[1, 7, 16, 20] focus on recommending suitable developers for code changes or PRs, either by calculating the similarity of developers' technical focus or generating features from the modified source code. However, they just focus on the precision, recall or accuracy values, the reviewers that they recommend still need to be analyzed.

The contributions of our work are as follows:

- We statistically analyze the review process in Rails, and find that inactive reviewers are important to the reviewing process. What's more, code reviewer recommendation is of great importance to the time-consuming process.
- We reproduce a popular and effective recommendation algorithm (IR-based recommendation) based on the large dataset of Rails containing 16,049 PRs. We justify that the existing recommendation approaches are easy

Internetware'17, September 23, 2017, Shanghai, China

ACM ISBN 978-1-4503-5313-7/17/09…$15.00
https://doi.org/10.1145/3131704.3131718

to get into overfitting trouble, i.e., excessive reviewing tasks tend to be recommended to active reviewers.

The rest of this paper is organized as follows. Section 2 reviews a few related studies. Section 3 describes the empirical study of PRs in Rails. Section 4 presents the IR-based reviewer recommendation method. Section 5 describes the experiment settings. Section 6 discusses the experiment results. Section 7 elaborates the conclusion and describe the future plans of the study.

2 RELATED WORK

Modern Code Review

Code review is widely-agreed as the best software engineering practice in both industrial and open source contexts [2, 9], which helps to improve the quality of source code and reduce the defects in open source repositories. Traditionally, code reviewing is a time-consuming and heavy-weight process, which requires experts and a group meeting most often [5]. In order to speed up the process and improve the effectiveness at the same time, there comes up many modern code review methods. McIntosh, Kamei et al. [10] evaluated the impact of code review coverage to the software quality. After that, they extended their work and studied the influence of different types of code reviews through an empirical study [11]. Bosu [3] and Morales [12] also found that code review did have impact to security vulnerabilities and design quality respectively. In addition to empirical studies, there are also many studies focusing on the development of code reviewing tools. For example, Mukadam et al. [13] developed Gerrit for reviewing android based source code.

Developer Recommendation

With the increase of developers and repositories in social coding communities, it is becoming more and more important to recommend developers to different tasks so that there can be a better development of open source repositories.

Bug assignment. There are large amount of bug reports and huge number of developers in popular open source projects (rails has more than 28K issues up to May 2017), which makes it a labor-intensive task to assign bugs to suitable fixers. Therefore, recommending fixers to bug reports is of significant importance. Jeong, Kim et al. [6] recommended bug fixers by building the tossing graph according to the bug fix history. Kagdi, Poshyvanyk et al. [8] matched bug reports with developers by extracting the technical terms of source code committed by contributors and bug report description. Canfora and Cerulo [4] also used the information retrieval (IR) method, and indexed developers through the textual information of bug reports.

Contributor recommendation. There are some other works focusing on recommending contributors across different projects in the whole community. Zhang et al. [23] expanded users' activities in social coding communities by matching users from both GitHub and StackOverflow. Then recommending potential developers who may have interests in the target project. After that, Zhang [22] also proposed a hybrid method by combing the weighted collaborative filtering algorithm and the text matching algorithm based on the collaborative network in GitHub.

Code reviewer recommendation. For the problem that we focus on, Jeong, Kim et al. [7] predicted the code reviewer by Bayesian Network. They trained the model using features generated from the source code and code files. Thongtanunam et al. [16] found code reviewers by comparing the file path. Balachandran [1] found suitable reviewers by checking the change history of source code lines. Rahman et al. [14] recommended code reviewers across different projects. Yu et al. [18–20] proposed methods for pull-based code reviewer recommendation based on the social comment network, which can reduce the human effort in reviewing code changes [17, 21].

3 EMPIRICAL STUDY

The goal of our work is to find the necessity of recommending code reviewers for PRs in Rails, and to evaluate the influence of active developers on the recommendation result. Because there are so many PRs, and code review is a time-consuming process which is based on the reviewers' ability. It is unpractical and unnecessary for all the developers to review all the PRs. In order to deeply understand the relationship between developers and PRs, and make preparations for the code reviewer recommendation model, we propose two research questions.

- RQ1: What is the difference between active and inactive code reviewers? Do PRs need inactive reviewers indeed?
- RQ2: Why we say that code review is a time-consuming process?

We do empirical studies on 16,049 PRs and 5,075 reviewers before May 2017 in Rails project.

RQ1

In order to answer RQ1, we firstly analyze the reviewer number of each PR to see whether PR reviewing process needs many reviewers. The results is shown in Figure 1.

From Figure 1, we can see that, the number of reviewers for each PR is very small. And the PRs which have 1 or 2 reviewers takes up 53.7%. Moreover, the number of PRs which have more than 10 reviewers only takes up 1.1%. That

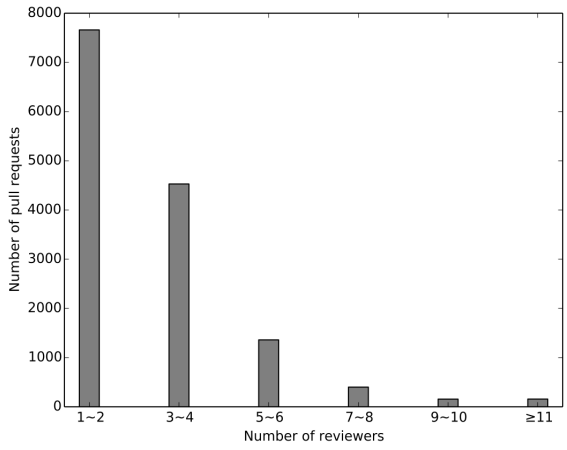

Figure 1: Number of Reviewers for Pull Requests

is to say, when reviewing a PR, it is not necessary to invite many reviewers.

Now that there is no need to invite so many reviewers to review a PR, will active reviewers undertake all the reviewing work? We calculate the number of reviewed PRs for each developer in Rails, and get the changing curve of the top 100 reviewers. The result is shown in Figure 2.

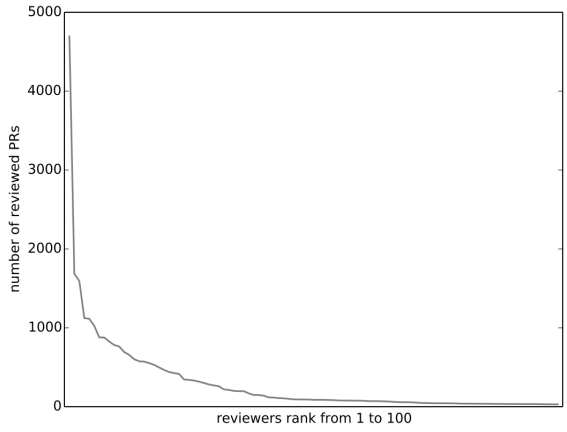

Figure 2: Number of Reviews for Top 100 Reviewers

Figure 2 shows that for the top 100 active reviewers in Rails, even though the number of reviewed PRs decreases sharply from 4,695 to 30, the inactive developers also undertake a large amount of reviewing work. That is to say, the difference between active and inactive developers is very big, however, inactive developers also play an important role.

We take *pixeltrix* as an example, whose activeness ranks the 13_{th} among all the code reviewers in Rails. He has reviewed 657 PRs till March 2017. From Figure 3, we see that *pixeltrix* also submit reviews of high quality.

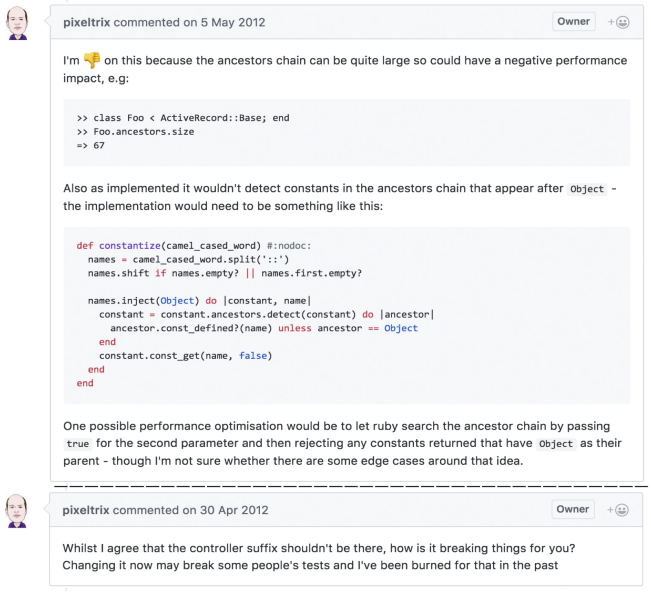

Figure 3: The Reviews by *pixeltrix*

From the above analysis, we conclude that although code review is led by some super active reviewers, other reviewers can also make a lot of contribution in the code reviewing process. Therefore, when a PR comes, we do not need to recommend those developers who are highly involved in the reviewing process. On the contrary, it is important to recommend other reviewers to undertake the code reviewing work in order to reduce the burden of those active reviewers and speed up the code reviewing process.

RQ2

For RQ2, in order to verify that code review is a time-consuming process, we firstly calculate the time used for closing PRs, which is shown in Figure 4.

Figure 4 shows that 40.4% PRs are closed in more than 10 days. Moreover, 28.9% PRs are closed more than a month. That is to say, it is a long time for reviewers to finish reviewing the code change. However, there are two possible reasons for this situation. One is the start time for reviewing the PR is late, the other is the reviewing process is time-consuming. Therefore, we calculate the standard deviation of the first three reviewers' start time for each PR, which is shown in Figure 5.

From Figure 5, we can see that for 68.8% PRs, the standard deviation of the first three reviewers' review time is less than

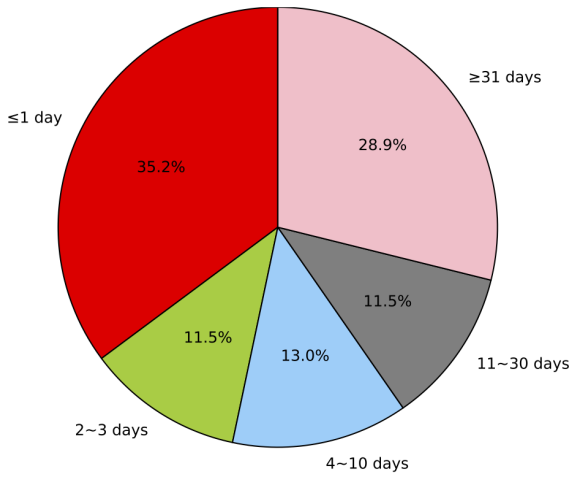

Figure 4: Time Used for Closing Pull Requests

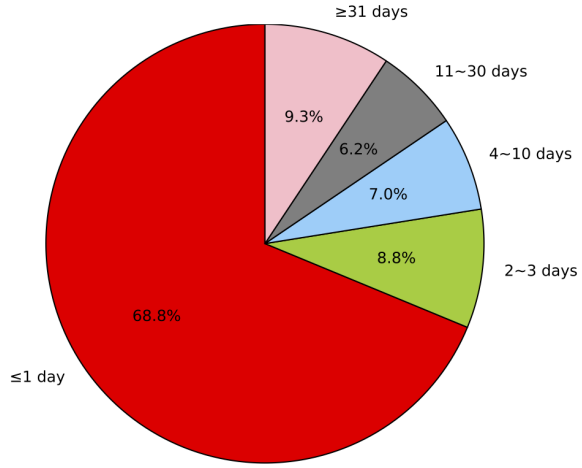

Figure 5: The Standard Deviation of the First Three Reviewers' Start Time

one day, which means that the time gap between each reviewer's participation is less than 1 day. Therefore, these PRs can attract reviewers to participate in a short time. However, there are still 32.2% PRs which cannot attract enough PRs in a short time. For them, it is important to apply code reviewer recommendation algorithm so that different reviewers can focus on them at the same time. For all the PRs, code reviewer recommendation can help to find suitable reviewers and shorten the time for reviewing theoretically.

4 REVIEWER RECOMMENDATION METHOD

IR-based Reviewer Recommendation

In this section, we will present a simple and popular reviewer recommendation algorithm. From Yu's work [20], we find that the IR-based approach is the most suitable traditional method for recommending code reviewers to PRs in GitHub. The process of this method is show as below.

Firstly, we generate the technical terms of each PR in the project based on its title and description. By removing the stop words pre-defined before this process, we get the technical focus of each PR. All the technical terms can form into a corpus.

Secondly, we generate the term vector of each PR according to the *TF-IDF* algorithm (equation 1), where different PRs' technical terms will get different values.

$$TF-IDF(t,pr) = \frac{freq(t,pr)}{\sum_{t' \in T_{pr}} freq(t',pr)} \times log(\frac{\sum_{pr' \in PR} \sum_{t' \in T_{pr'}} freq(t',pr')}{\sum_{pr' \in PR} freq(t,pr')}) \quad (1)$$

In this equation, t represents a specific term. $freq(t,pr)$ means the number of times term t occurs in pull request pr. T_{pr} represents all the technical terms in pr.

Thirdly, when a test PR comes, we calculate the relationship between each PR using the cosine similarity (equation 2). The vector of each PR is formed by the technical terms in the whole corpus.

$$similarity(pr,pr') = \frac{v_{pr} \cdot v_{pr'}}{|v_{pr}||v_{pr'}|} \quad (2)$$

After that, we summarize the relationship between each PR that a developer has reviewed before and the target PR. Then we get the relationship between the developer and the target PR (equation 3).

$$relation(d,pr) = \sum_{pr' \in PR_d} similarity(pr,pr') \quad (3)$$

In this equation, d represents a developer. PR_d means the PR set that the developer d has reviewed before.

Therefore, when a target PR comes, we calculate the relationship between each developer and the target project. Then rank the values in descending order, and recommend the top several results.

Rails Robot

For Rails project itself, there is a developer called "rails-bot" which was created on May 2_{th}, 2014. It is actually a robot which is used to assign code reviewers for Rails PRs. When a PR comes, "rails-bot" can find suitable developers for the PR and make a comment below by "@" someone to review.

This robot is designed to help reviewers find related PRs and speed up the reviewing process.

5 EXPERIMENT

Data set

In this section, we will present the data set that we use to test the IR-based recommendation model. We focus on the recommendation of PRs in Rails [1] in GitHub.

For the performance test of IR-based recommendation method itself, we select the PRs from January, 2012 to February, 2017, and get 16,049 in total. These PRs are used to form the term corpus of IR-based recommendation algorithm. Among these PRs, we select those created from February 2016 to February 2017, then filtering those with no actual reviewers [2]. Finally, we get 3,034 PRs for the test process.

For the performance comparison between IR-based recommendation approach and Rails Robot, we select the PRs within the test set in the first experiment that Rails Robot has made recommendation, and finally get 2,225 PRs.

The data set can be seen in table 1.

Table 1: Dataset In GitHub

Experiment	Process	PR Number
IR-based Method Performance	Training Corpus	16,049
	Test	3,034
Performance Comparison	Training Corpus	16,049
	Test	2,225

Experiment Metrics

For the validation of our experiments, precision and recall are not suitable to measure the model's performance. For the false positives of our recommendation result (the recommended developers who have not reviewed the PR), they can still review the PR sometime in the future. Even though the PR is closed, it can still be reopened. Therefore, the precision and recall values will change according to the time frame that we select.

Therefore, we use the accuracy value to measure the performance of our recommendation model. The equation is shown as below (equation 4).

$$accuracy = \frac{\sum_{i=1}^{|PR_s|} hasTrue(PR_{s_i})}{PR_s} \quad (4)$$

where *PRs* means the set of test PRs in the target project, and hasTrue is a function which checks whether a PR has got a true code reviewer.

[1] https://github.com/rails/rails

[2] actual reviewers: reviewers that are not the PR author or the Rails Robot.

6 RESULTS

In this section, we will present the results of the two experiments, the performance of IR-based recommendation algorithm and the comparison between IR-based recommendation algorithm and Rails Robot.

Firstly, in order to check the effectiveness for recommending those inactive reviewers, we remove those super active reviewers to see the accuracy value. The result can be seen in Figure 6.

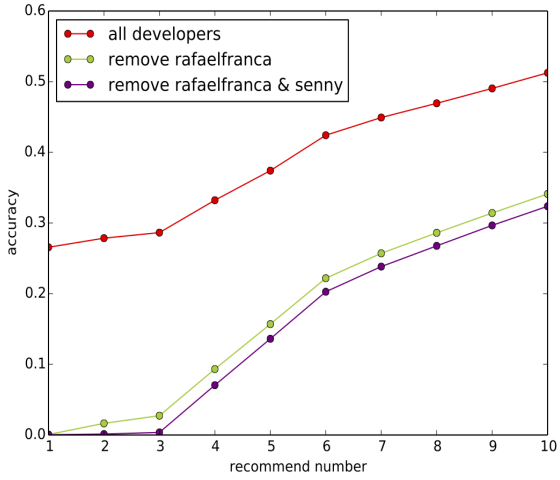

Figure 6: Accuracy of IR-based Recommendation Algorithm

From Figure 6, we can see that when recommending 1 to 10 reviewers, the accuracy value of the IR-based algorithm ranges from 26.6% to 51.3%. However, when we remove the most active reviewer called "rafaelfranca", the accuracy drops a lot, ranging from 0.07% to 34.1%. Moreover, when we remove the top 2 reviewers called "rafaelfranca" and "senny", the accuracy will also drop to some extent. That is to say, the IR-based recommendation algorithm is influenced by the activeness of reviewers and is tend to recommend active developers to review PRs, however, these reviewers will participate in the reviewing process spontaneously. Therefore, a more effective recommendation algorithm that focuses on the inactive reviewers is still in demand.

Although the IR-based recommendation algorithm still needs to be improved, comparing to the Rails Robot, it performs better. When recommending one developer to the 2,225 PRs, the accuracy for IR-based Recommendation Algorithm is 25.1%, however the accuracy for Rails Robot is 18.7%.

7 CONCLUSIONS AND FUTURE WORK

This study explores the pull-request review process of Rails in GitHub intensively, and discusses the weakness of the current popular recommendation algorithms. We firstly did an empirical study and conclude that code reviewer recommendation especially for those inactive reviewers is very important. Then we test the IR-based recommendation algorithm and found that this method mainly recommends those active reviewers, which is not that useful for reviewer recommendation. However, comparing with the Rails Robot for Rails project itself, it still performs better.

For the future work, we will firstly create a more effective code reviewer recommendation algorithm based on the behavior of inactive developers and then do some fine-grained recommendation work. We will check reviewers' different types of participation, including the code layer review and the management layer review. Then based on PRs' different requirements, we will recommend different types of reviewers to speed up the reviewing process.

ACKNOWLEDGEMENTS

The research is supported by the National Grand R&D Plan (Grant No. 2016-YFB1000805) and National Natural Science Foundation of China (Grant No.61502512, 61432020, 61472430, 61532004). We would like to thank Tao Wang and Yue Yu for their collaboration.

REFERENCES

[1] Vipin Balachandran. 2013. Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In *Software Engineering (ICSE), 2013 35th International Conference on*. IEEE, 931–940.

[2] Barry Boehm and Victor R Basili. 2005. Software defect reduction top 10 list. *Foundations of empirical software engineering: the legacy of Victor R. Basili* 426 (2005), 37.

[3] Amiangshu Bosu and Jeffrey C Carver. 2013. Peer code review to prevent security vulnerabilities: An empirical evaluation. In *Software Security and Reliability-Companion (SERE-C), 2013 IEEE 7th International Conference on*. IEEE, 229–230.

[4] Gerardo Canfora and Luigi Cerulo. 2006. Supporting change request assignment in open source development. In *Proceedings of the 2006 ACM symposium on Applied computing*. ACM, 1767–1772.

[5] Michael E Fagan. 2001. Design and code inspections to reduce errors in program development. In *Pioneers and Their Contributions to Software Engineering*. Springer, 301–334.

[6] Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. 2009. Improving bug triage with bug tossing graphs. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, 111–120.

[7] Gaeul Jeong, Sunghun Kim, Thomas Zimmermann, and Kwangkeun Yi. 2009. 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)* (2009), 1–18.

[8] Huzefa Kagdi and Denys Poshyvanyk. 2009. Who can help me with this change request?. In *Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on*. IEEE, 273–277.

[9] Sami Kollanus and Jussi Koskinen. 2009. Survey of software inspection research. *The Open Software Engineering Journal* 3, 1 (2009), 15–34.

[10] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. 2014. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 192–201.

[11] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. 2016. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering* 21, 5 (2016), 2146–2189.

[12] Rodrigo Morales, Shane McIntosh, and Foutse Khomh. 2015. Do code review practices impact design quality? a case study of the qt, vtk, and itk projects. In *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*. IEEE, 171–180.

[13] Murtuza Mukadam, Christian Bird, and Peter C Rigby. 2013. Gerrit software code review data from android. In *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*. IEEE, 45–48.

[14] Mohammad Masudur Rahman, Chanchal K Roy, and Jason A Collins. 2016. CoRReCT: code reviewer recommendation in GitHub based on cross-project and technology experience. In *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM, 222–231.

[15] Peter C Rigby and Margaret-Anne Storey. 2011. Understanding broadcast based peer review on open source software projects. In *Proceedings of the 33rd International Conference on Software Engineering*. ACM, 541–550.

[16] 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. In *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*. IEEE, 141–150.

[17] Yue Yu, Huaimin Wang, Vladimir Filkov, Premkumar Devanbu, and Bogdan Vasilescu. 2015. Wait For It: Determinants of Pull Request Evaluation Latency on GitHub. In *MSR*.

[18] Yue Yu, Huaimin Wang, Gang Yin, and Charles X Ling. 2014. Reviewer recommender of pull-requests in GitHub. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*. IEEE, 609–612.

[19] Yue Yu, Huaimin Wang, Gang Yin, and Charles X Ling. 2014. Who should review this pull-request: Reviewer recommendation to expedite crowd collaboration. In *Software Engineering Conference (APSEC), 2014 21st Asia-Pacific*, Vol. 1. IEEE, 335–342.

[20] Yue Yu, Huaimin Wang, Gang Yin, and Tao Wang. 2016. Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment? *Information and Software Technology* 74 (2016), 204–218.

[21] Yue Yu, Gang Yin, Tao Wang, Cheng Yang, and Huaimin Wang. 2016. Determinants of pull-based development in the context of continuous integration. *Science China Information Sciences* 59, 8 (2016), 080104.

[22] Xunhui Zhang, Tao Wang, Gang Yin, Cheng Yang, and Huaimin Wang. 2017. Who Will be Interested in? A Contributor Recommendation Approach for Open Source Projects. In *Proceedings of the 29th International Conference on Software Engineering & Knowledge Engineering Accepted*.

[23] Xunhui Zhang, Tao Wang, Gang Yin, Cheng Yang, Yue Yu, and Huaimin Wang. 2017. DevRec: A Developer Recommendation System for Open Source Repositories. In *International Conference on Software Reuse*. Springer, 3–11.