Reviewer Recommender of Pull-Requests in GitHub

Yue Yu*, Huaimin Wang*, Gang Yin*, Charles X. Ling†
*National Laboratory for Parallel and Distributed Processing,
College of Computer, National University of Defense Technology, Changsha, 410073, China
†Department of Computer Science, The University of Western Ontario, London, Ontario, Canada N6A 5B7
{yuyue,hmwang,yingang}@nudt.edu.cn, cling@csd.uwo.ca

***Abstract*—Pull-Request (PR) is the primary method for code contributions from thousands of developers in GitHub. To maintain the quality of software projects, PR review is an essential part of distributed software development. Assigning new PRs to appropriate reviewers will make the review process more effective which can reduce the time between the submission of a PR and the actual review of it. However, reviewer assignment is now organized manually in GitHub. To reduce this cost, we propose a reviewer recommender to predict highly relevant reviewers of incoming PRs. Combining information retrieval with social network analyzing, our approach takes full advantage of the textual semantic of PRs and the social relations of developers. We implement an online system to show how the reviewer recommender helps project managers to find potential reviewers from crowds. Our approach can reach a precision of 74% for top-1 recommendation, and a recall of 71% for top-10 recommendation.** `http://rrp.trustie.net/`

***Keywords*—Pull-request, Reviewer Recommendation, Social Network Analysis, Distributed Software Development**

I. Introduction

GitHub[1], a popular social coding community [1], attracts a large number of software projects hosted on it. Pull-Request (PR) is the primary method [2], [3] for code contributions from thousands of developers. Currently, it is not uncommon in the popular projects to receive tens of PRs daily covering nearly 60% of code commits from contributors.

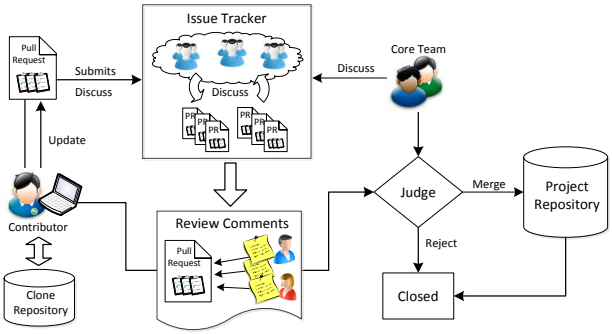


Figure 1. The overview of pull-request mechanism

The overview of PR mechanism is presented as Figure 1. Firstly, a contributor implements some new features or fixes bugs based on his personal repository cloned from the latest version of project repository. When his work is finished, the patches are packaged as a PR submitted to the issue tracker. The system open a new issue for this PR, and then add the issue to an awaiting list to be reviewed. In GitHub, the traditional review process is transformed into a crowdsourcing job. Not only core developers but also external developers in the community can act as reviewers. The reviewers can freely discuss the PR with the contributor and core developers in terms of their interests and expertise. Next, in the light of reviewers' suggestions, the contributor would update his pull-request by attaching new commits, and then reviewers discuss that PR again. Finally, the responsible mangers of the core team take all the opinions of reviewers into consideration, and then merges or rejects that PR. Thus, we can see PR review is an important way to maintain the quality of the software.

Assigning incoming PRs to appropriate reviewers will make PR review more effective, because it can reduce the time between the submission of a PR and the actual review of it. We refer to the period between the time when a PR is submitted into issue tracker and the time when it begins to be discussed by reviewers as review latency. The PRs which have been assigned to reviewers have lower review latency than those without assignment. According to our analysis, the time of the recommended reviewer submitting his first comment on PR is on average 40.8 hours shorter than those without recommendation. However, reviewer assignment is now organized manually in GitHub. As each developer in community has the chance to join the review discussions, the project managers may not completely find out all potential reviewers from crowds.

To reduce this cost, we designed a reviewer recommender to predict appropriate reviewers for incoming PRs in Github. The two key intuitions of our approach focus on the textual semantic of PRs and the social relations of contributors.

- The expertise of a reviewer can be learned from his PR-commenting history. For a newly received PR, the developers who have commented similar PR frequently in the past are the suitable candidates to review the new one.
- The common interests among developers can be measured by social relations between contributors and reviewers in historical PRs. The developers who share more common interests with the contributor are the appropriate reviewers of his incoming PRs.

Thus, we propose a novel approach combining information retrieval with social network analyzing to recommend highly relevant reviewers. We demonstrated the efficiency of our approach on 10 popular projects which have received over

[1]https://github.com/

1000 PRs in GitHub. On average, our reviewer recommender can reach a precision of 74% on the top-1 recommendation and a recall of 71% on top-10 recommendation.

II. Exiting Tools and Related work

A. Exiting Tools and Motivation

Notifications in Github are based on the repositories you are watching. Each project watcher will mechanically receive notifications of all newly received PRs. In this way, the PRs which the developer really cares about would be drowned out by a massive amount of noise.

To triage PRs, the project managers can use a label to assign a new PR to one of core developer. The assignee is in charge of the review process. However, only 0.89% of PRs have been set that label in our dataset. Besides, the *@mention* tool is widely used in the discussion of PR review. If a @ symbol placed in front of a user's ID, the corresponding developer will receive a special notification that he has been mentioned in the PR and his ID would be highlighted.

Figure 2. An example of discussion among reviewers in a pull-request

Taking a real PR of *Ruby on Rails* as an example, as shown in Figure 2, no one is assigned to this PR. A core developer called *rafaelfranca* is the first reviewer to comment the PR. As he considers that *javan*'s work would be relevant, he mentions (@) *javan* to join the discussion. At the second post, we can see that *javan* indeed presents his opinion. Apart from *javan* who is informed by *rafaelfranca*, other three reviewers comment that PR spontaneously. Because all the comments affect the decision-making of that PR, if they do not catch it timely, it is possible that some vital opinions would be lost.

Thus, if appropriate reviewers are automatically recommended when a new pull-request is submitted, the review process would be more efficient. It is worth mentioning that the novel application of reviewer recommender can be seamlessly integrated with the social media of peer-to-peer notification, such as the *@mention* tool in GitHub that adds the @ tags to the potential reviewers automatically. Actually, we also can provide an independent service for each OSS community, when the enough data has been shared and collected.

B. Related Work

We review the similar researches of reviewer recommendation in bug triaging and code (patch) review in this section.

The most representative researches are based on machine learning and information retrieval techniques to triage incoming bug reports. For example, Anvik et al. [4] apply a machine learning algorithm to learn the kinds of reports each developer resolves and recommend developers for a new bug report. Kagdi and Poshyvanyk [5], [6] extract the comments and identifiers from the source code and index these data by Latent Semantic Indexing (LSI). For a new bug report, such indexes can be useful to identify the most appropriate developers to resolve it. To support code review, Balachandran [7] designs a tool called *Review Bot* to predict the developers who have modified related code sections frequently as appropriate reviewers. Compared to the *Review Bot*, Thongtanunam et al. [8] recommends code reviewers from developers who have examined files with similar directory paths.

All these approaches focus on mining the related text, such as bug descriptions and code files, but the social relations among developers are ignored. However, the process of PR review is more likely to rely on the discussions among reviewers to be resolved. These discussions occurred over artifacts may imply reviewers common interests in social activities and division of work in projects. Thus, we consider the relations reflected by the artifact-mediated communication is a key factor of recommendation. Begel et al. [9] present a framework for connecting developers and their work artifacts together. In this paper, we propose a novel and lightweight approach that combines information retrieval with social network analyzing to recommend appropriate reviewers for new PRs.

III. Methodology

Firstly, the titles and descriptions of PRs are extracted and indexed using the Vector Space Model. Then, we measure semantic similarity between the new PR and each one of historical PRs, and predict the expertise score of a developer according to the number of comments he has submitted. Furthermore, we construct a comment network for each project separately by analyzing the comment relations among developers. In a specific project, we can predict the common interest of each reviewer shared with the PR contributor based on the comment network. Finally, we synthesize the expertise scores and the common interests to rank all candidates.

A. Vector Space Model of Pull-Request

Each PR is characterized by its title and description, and labeled with a set of names about developers who had submitted at least one comment to it. Then, all stop words and non-alphabetic tokens are removed, and remaining words are

stemmed. We use the vector space model to represent each PR as a weighted vector. Each element in the vector is a term, and the value stands for its importance for the PR. Term frequency-inverse document frequency (tf-idf) is utilized to indicate the value of a term, which can be calculated as Equation 1.

$$tfidf(t, p_r, P_R) = log(\frac{n_t}{N_{p_r}} + 1) \times log \frac{N_{P_R}}{|p_r \in P_R : t \in p_r|} \quad (1)$$

where t is a term, p_r is a pull-request, P_R is the corpus of all pull-requests in a given project, n_t is the count of appearance for term t in p_r, and N_{p_r} and N_{P_R} are the total number of terms in p_r and pull-requests in corpus respectively.

B. Semantic Similarity and Expertise Score

For a newly recieved PR, we firstly retrieve top-k relevant PRs from our corpus. We use *cosine similarity* to measure the semantic similarity between a new PR and each of the resolved PRs. We consider that the more frequency the reviewer has commented relevant PRs, the more knowledgeable that reviewer is in handling the new PR. Therefore, the expertise score of a reviewer can be predicted based on the number of comments which they has published in the top-k relevant PRs (i.e., cosine similarity score times the number of comments).

C. Comment Network and Common Interest

We consider that common interests among developers are *project-specific*, so we build a *comment network* for each project separately. In a given project, the structure of comment relations is a many-to-many model. There are many contributors have submitted PRs to a project, and a contributor can be a reviewer to comment other contributors' PRs. A PR would be commented by several reviewers more than once.

The *comment network* is defined as a weighted directed graph $G_{cn} = \langle V, E, W \rangle$, where the set of developers is indicated as vertices V and the set of relations between nodes as edges E. If node v_j has reviewed at least one of v_i's PRs, there is a edge e_{ij} from v_i to v_j. The set of weights W reflects the importance degree of edges, and the weight w_{ij} of e_{ij} can be evaluated by Equation 2.

$$w_{ij} = \sum_{r=1}^{k} w_{(ij,r)} = P_c \times \sum_{r=1}^{k} \sum_{n=1}^{m} \lambda^{n-1} \times t_{(ij,r,n)} \quad (2)$$

where k is the total number of PR submitted by v_i, and $w_{(ij,r)}$ is a component weight related to an individual PR r. P_c is an empirical default value[2] (set to 1.0), which is reserved to estimate the influence of each comment on the PR, and m is the sum of comments submitted by v_j in the same PR r. When reviewer v_j published multiple comments ($m \neq 1$) in the same PR, his influence is controlled by a decay factor λ (set to 0.8). The element $t_{(ij,r,n)}$ is a time-sensitive factor of corresponding comment which can be calculated as below:

$$t_{(ij,r,n)} = \frac{timestamp_{(ij,r,n)} - baseline}{deadline - baseline} \in (0, 1] \quad (3)$$

[2]User comments can be found in pull-requests, issue posts and commit files. Here, we just use the comments of pull-request.

where $timestamp_{(ij,r,n)}$ is the date that reviewer v_j presented the comment n in the PR r which is reported by contributor v_i. The $baseline$ and $deadline$ are highly related to the selection of training set. If we use the data from 2012-01-01 to 2013-05-31 to learn the weights, the parameters $baseline$ and $deadline$ are set to 2011-12-31 and 2013-05-31 respectively.

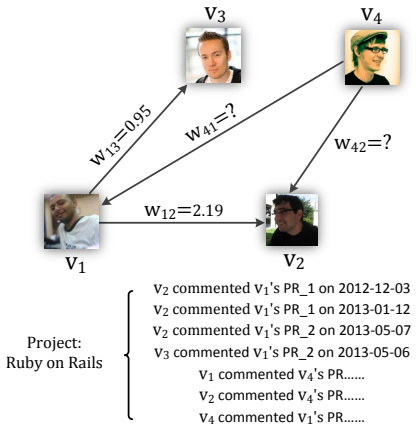


Figure 4. An example of the *comment network*

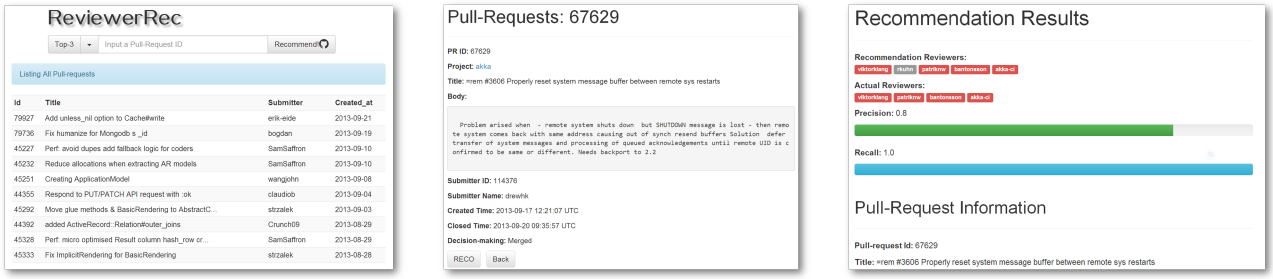
Figure 4 shows an example of a part of *comment network* about *Ruby on Rails*. Two different PRs (`PR_1 and PR_2`) reported by v_1 have been commented by v_2 and v_3, so there are two edges from v_1 to v_2 and v_1 to v_3. Evaluating the relation between v_1 and v_2, k of Equation 2 equals 2, because v_2 reviewed both two pull-requests. For `PR_1`, v_2 commented it twice, so we set $m = 2$. The first time-sensitive factor of the date 2013-12-03 can be computed by Equation 3 that $t_{(12,1,1)} \approx 0.654$. In addition, at the date of 2013-01-12 ($t_{12,1,2} \approx 0.731$), another review published by v_2 in `PR_1` should be controlled by λ (set to 0.8) due to the diminishing impact of one user in the same PR, so $w_{(12,1)}$ can be calculated as: $P_c \times (t_{(12,1,1)} + \lambda^{2-1} \times t_{(12,1,2)}) \approx 1.24$. Similarly, the weight $w_{12} = w_{(12,1)} + w_{(12,2)} = 2.19$, and $w_{13} = 0.95$. Thus, we can predict that reviewer v_2 share more common interests with contributor v_1 compared with v_3, which has been quantified by the corresponding weights of edges.

The *comment network* has several desirable qualities.

- The global collaboration structure is revealed between contributors and reviewers in a given project.
- The time-sensitive factor t is introduced to guarantee that the recent comments are more valuable for the weights of edges than the old comments.
- The decay factor λ is introduced to guarantee the difference values between the comments submitted to multiple PRs or single PR. For example, if reviewer v_j commented 5 different PRs of v_i and meanwhile v_q commented one of v_i's PRs 5 times, the weight of w_{ij} is larger than w_{iq}.

D. Reviewer Recommendation

When a contributor submits a new PR, we use the PR text to calculate expertise scores of candidates. Then, their common interests can be calculated by starting from the contributor in the *comment network*. In this paper, we regard that the factor of common interest is as important as the factor of expertise



(a) Homepage of a project (b) Summary of a PR (c) Recommendation Result

Figure 3. The online system of reviewer recommender

in each project. Thus, we standardize the factors, and then add them together to recommend top-k reviewers to the new PR. In future, we plan to deeply analyze the influence of each factor exerted on different projects.

IV. Reviewer Recommender and Experiment

We implement an online system of reviewer recommender with B/S architecture. It consists of two parts: a backend server that processes data and a web interface that interacts with the user. In practice, our application would be the most effectively used as a plug-in for social coding communities.

On the backend, the server can continuously collect and analyze new PRs from GitHub. When a project receives a new PR, it will be listed in its homepage (Figure 3 (a)). We extract the key information of a PR such as the ID, description and submitter, as shown in Figure 3 (b). Project managers can pick an new PR from the awaiting list or submit a query for specific PR by typing in the issue ID. The number of recommendation can be set as required. Then, in Figure 3 (c), the predicted result will be shown together with the measures. We evaluate the performances of our approaches using precision and recall which are widely used as standard metrics in previous work. In that case, top-5 reviewers have been recommended, and 4 of them have actually commented that PR in reality. Hence, the precision is 80% and the recall is 100%.

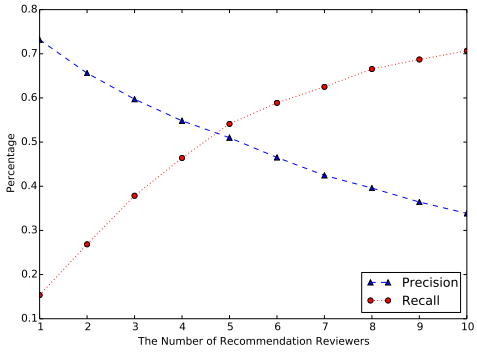


Figure 5. Precision vs. Recall of reviewer recommender

In this paper, we demonstrate the performances of our approach on 10 popular projects which have received over 1000 pull-requests. We use the data from 2012-01-01 to 2013-05-31 as the training set and the data from 2013-06-01 to 2013-10-01 as the test set. Figure 5 clearly exhibits the overall performance of our approach. On average, the precision reaches the highest point of 0.74 for top-1 recommendation and the recall considerably ascends to 0.71 at the point of top-10 recommendation. It means we can successfully hit a majority of reviewer. Especially, high precision of top-1 recommendation is significant. As the example of Figure 2, if the first reviewer can be predicted, he would remind other reviewers to joint the discussion with *@mentioning*.

V. Conclusion and Future Work

For a new PR, recommending reviewers will make the review process more effective. In this paper, we propose a novel approach that combine information retrieval with social network analyzing. In the future, we plan to explore how to use other types of social networks, such as the *watcher network*, to improve the performance of our method.

VI. Acknowledgement

This research is supported by the National High Technology Research and Development Program of China (Grant No. 2012AA011201).

References

[1] A. Begel, J. Bosch, and M.-A. Storey, "Social networking meets software development: Perspectives from github, msdn, stack exchange, and topcoder," *IEEE Software*, vol. 30, no. 1, pp. 52–66, 2013.

[2] J. Tsay, L. Dabbish, and J. Herbsleb, "Influence of social and technical factors for evaluating contribution in github," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE '14, 2014, pp. 356–366.

[3] G. Gousios, M. Pinzger, and A. v. Deursen, "An exploratory study of the pull-based software development model," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE '14. New York, NY, USA: ACM, 2014, pp. 345–355.

[4] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th International Conference on Software Engineering*, ser. ICSE '06. New York, NY, USA: ACM, 2006, pp. 361–370.

[5] H. Kagdi and D. Poshyvanyk, "Who can help me with this change request?" in *Program Comprehension, 2009. ICPC '09. IEEE 17th International Conference on*, May 2009, pp. 273–277.

[6] M. Linares-Vasquez, K. Hossen, H. Dang, H. Kagdi, M. Gethers, and D. Poshyvanyk, "Triaging incoming change requests: Bug or commit history, or code authorship?" in *Software Maintenance, 28th IEEE International Conference on*, Sept 2012, pp. 451–460.

[7] V. Balachandran, "Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13, 2013, pp. 931–940.

[8] P. Thongtanunam, R. G. Kula, A. E. C. Cruz, N. Yoshida, and H. Iida, "Improving code review effectiveness through reviewer recommendations," in *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering*, ser. CHASE '14, 2014, pp. 119–122.

[9] A. Begel, Y. P. Khoo, and T. Zimmermann, "Codebook: Discovering and exploiting relationships in software repositories," in *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10, 2010, pp. 125–134.