Who Should Review This Pull-Request: Reviewer Recommendation to Expedite Crowd Collaboration

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*—Github facilitates the pull-request mechanism as an outstanding social coding paradigm by integrating with social media. The review process of pull-requests is a typical crowdsourcing job which needs to solicit opinions of the community. Recommending appropriate reviewers can reduce the time between the submission of a pull-request and the actual review of it. In this paper, we firstly extend the traditional *Machine Learning* (ML) based approach of bug triaging to reviewer recommendation. Furthermore, we analyze social relations between contributors and reviewers, and propose a novel approach to recommend highly relevant reviewers by mining *comment networks* (*CN*) of given projects. Finally, we demonstrate the effectiveness of these two approaches with quantitative evaluations. The results show that *CN*-based approach achieves a significant improvement over the ML-based approach, and on average it reaches a precision of 78% and 67% for top-1 and top-2 recommendation respectively, and a recall of 77% for top-10 recommendation.**

***Keywords*—Pull-request, Reviewer Recommendation, Comment Network, Social Coding**

I. Introduction

The pull-based software development model [1], compared to traditional methods such as email-based patching [2] [3], makes developers contribute to software projects more flexibly and efficiently. In GitHub, the pull-request mechanism is upgraded to a unique social coding paradigm [4] by integrating multiple social media involving *follow*, *watch*, *fork* and *issue tracker*. The socialized pull-request model is pushing software evolution and maintenance into crowd-based development [5].

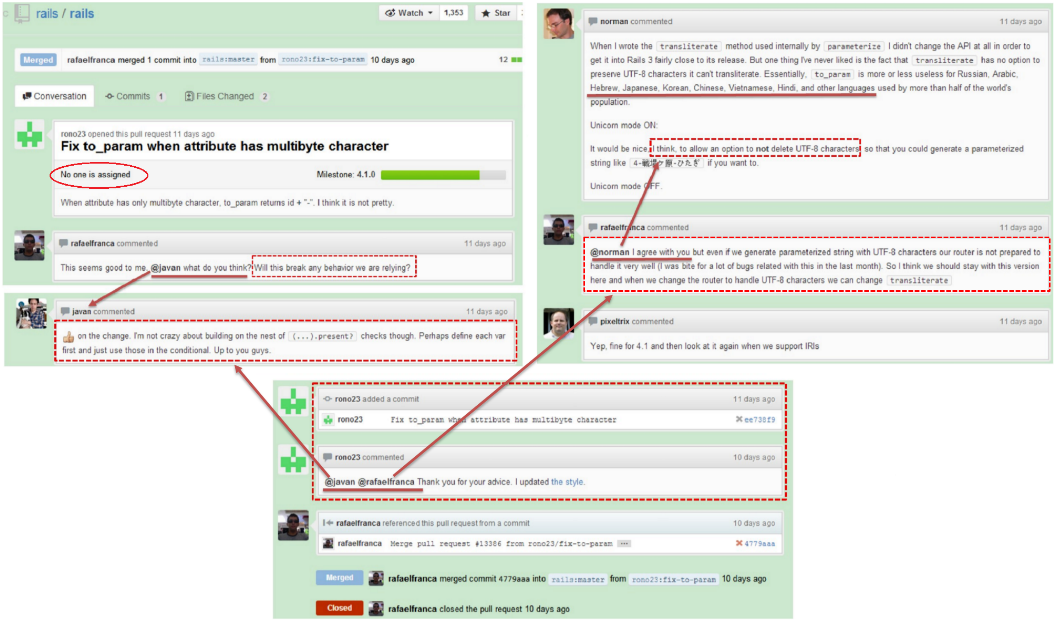
A typical contribution process [6] in GitHub involves following steps. First of all, a contributor could find an attractive project by following some well-known developers and watching their projects. Secondly, by forking an interesting one, the contributor implements a new feature or fixes some bugs based on his cloned repository. When his work is finished, the contributor sends the patches from the forked repository to its source by a pull-request. Then, all developers in the community have the chance to review that pull-request in the issue tracker. They can freely discuss whether the project needs that feature, whether the code style meets the standard or how to further improve the code quality. Next, in the light of reviewers' suggestions, the contributor would update his pull-request by attaching new commits, and then reviewers discuss that pull-request again. Finally, a responsible manager of the core team takes all the opinions of reviewers into consideration, and then merges or rejects that pull-request.

As a mushrooming number of developers use the pull-request mechanism to contribute their ideas and suggestions in GitHub, the efficiency of software evolution and maintenance is highly related to the crowd-based review process. However, the discussion among reviewers is time-consuming. Some relevant reviewers may not notice the new pull-request in time. Recommending reviewer will make the review process more effective, because it can reduce the time between the submission of a pull-request and the actual review of it.

A pull-request contains a title and description summarized its contributions of bug fixes or feature enhancements, so it is similar to a bug report in bug tracking systems. To the best of our knowledge, there is very few studies of reviewer recommendation for pull-requests. The most similar researches [7]–[12] are the approaches for recommending developers with the right implementation expertise to fix incoming bugs. For a newly received bug report, these approaches firstly find out some similar historical reports or source code files by measuring text similarity. Then, the expertise of a developer can be learned based on the bug-fixing history, source revision commits or code authorship. If we only focus on the text of pull-requests, these approaches can be extended to assign pull-requests to appropriate developers. However, the social relations between pull-request contributors and reviewers are neglected. Compared to bug fixing, the review process of pull-request is a social activity depending on the discussions among reviewers in GitHub. Thus, social relation is one of key factors of reviewer recommendation.

In this paper, we firstly implement the *Machine Learning* (ML) based approach of Anvik et al. [7], which is one of the most representative work of bug triaging. Furthermore, we analyze social relations among reviewers and contributors, and propose a novel approach of reviewer recommendation. Central to our approach is the use of a novel type of social network called *Comment Network* (*CN*), which can directly reflect common interests among developers. Finally, we conduct an empirical study on 10 projects which have received over 1000 pull-requests in GitHub. As there is no previous work of reviewer recommendation for pull-requests, we design a simple and effective method as a comparison baseline in experiments. The quantitative evaluations show that our *CN*-based approach achieves significant improvements over the baseline and ML-based method.

1530-1362/14 $31.00 © 2014 IEEE
DOI 10.1109/APSEC.2014.57



rafaelfranca merged 1 commit into `rails:master` from `rono23:fix-to-param` 10 days ago

rono23 opened this pull request 11 days ago

Fix to_param when attribute has multibyte character

No one is assigned

Milestone: 4.1.0

When attribute has only multibyte character, to_param returns id + "-". I think it is not pretty.

rafaelfranca commented 11 days ago

This seems good to me. @javan what do you think? Will this break any behavior we are relying?

javan commented 11 days ago

👍 on the change. I'm not crazy about building on the nest of `(...).present?` checks though. Perhaps define each var first and just use those in the conditional. Up to you guys.

norman commented 11 days ago

When I wrote the `transliterate` method used internally by `parameterize` I didn't change the API at all in order to get it into Rails 3 fairly close to its release. But one thing I've never liked is the fact that `transliterate` has no option to preserve UTF-8 characters it can't transliterate. Essentially, `to_param` is more or less useless for Russian, Arabic, Hebrew, Japanese, Korean, Chinese, Vietnamese, Hindi, and other languages used by more than half of the world's population.

Unicorn mode ON:

It would be nice, I think, to allow an option to **not** delete UTF-8 characters, so that you could generate a parameterized string like `4-戦場ヶ原-ひたぎ` if you want to.

Unicorn mode OFF.

rafaelfranca commented 11 days ago

@norman I agree with you but even if we generate parameterized string with UTF-8 characters our router is not prepared to handle it very well (I was bite for a lot of bugs related with this in the last month). So I think we should stay with this version here and when we change the router to handle UTF-8 characters we can change `transliterate`

pixeltrix commented 11 days ago

Yep, fine for 4.1 and then look at it again when we support IRIs

rono23 added a commit 11 days ago

rono23 `Fix to_param when attribute has multibyte character` ee738f9

rono23 commented 10 days ago

@javan @rafaelfranca Thank you for your advice. I updated the style.

rafaelfranca referenced this pull request from a commit 10 days ago

rafaelfranca `Merge pull request #13386 from rono23/fix-to-param` 4779aaa

Merged **rafaelfranca** merged commit 4779aaa into `rails:master` from `rono23:fix-to-param` 10 days ago

Closed **rafaelfranca** closed the pull request 10 days ago

Figure 1. An example of discussion among reviewers in a pull-request

The remainder of this paper is structured as follows. Section II conducts an empirical study of pull-request and depicts a motivating example for recommending reviewers to incoming pull-requests. Section III present how to assign a pull-request to reviewers using ML technique, and Section IV propose our *CN*-based recommendation approach. Experiments and analysis can be found in Section V. Finally, we present related work in Section VI and draw our conclusions in Section VII.

II. Empirical Study of Pull-request

In this section, we firstly investigate the popularity of pull-request model in GitHub. Then, a typical process of discussion among reviewers is introduced with an example.

A. Popularity of Pull-request

Gousios et al. [1] has illustrated the popularity of pull-based development model based on the comparison of usage between pull-request and shared repository in GitHub. They draw a conclusion that pull-request is a significant mechanism for distribute software development, even though only 14% of repositories are using it in GitHub until February 2013.

In the current, many competitive projects are growing fast supported by pull-request such as *Ruby on Rails*[1] which has received more than 10000 pull-requests. Therefore, we further investigate the usage of pull-request model among projects which receive at least 100 pull-requests. There are 3587 projects (exclude forked repositories) are extracted from the latest database of GHTorrent [13]. These projects cover 53 different program languages, but the distribution of the number of projects is highly skewed. Top-5 program languages contain 67% of projects (JavaScript 678, Ruby 475, Python 460, Java 424 and PHP 362). From June 2011 to March 2014, the absolute number of new pull-requests increased dramatically reaching the peak of 76673 per month (Figure 2). Thus, we can see that a growing number of developers contribute to open source using the pull-request mechanism.

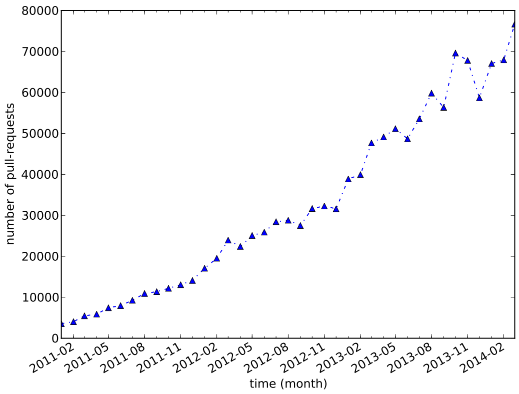


Figure 2. The growth of Pull-request quantity

B. Discussion among Reviewers

When a new pull-request arrives, the decision-making is a crowdsourcing process which depends on the discussion among reviewers. Taking a real pull-request of *Ruby on Rails* as an example, as shown in Figure 1, a core developer called *rafaelfranca* is the first reviewer to discuss this pull-request. As he thought that *javan*'s work would be relevant to this pull-request, he mentioned (@) *javan* to join the discussion. At the second post, we see that *javan* indeed presented his opinion. Meanwhile, other users participated in the discussion and made some important suggestions as well. Later, the author updated the pull-request by appending a new commit taking into account the above suggestions, and then he mentioned the two key reviewers for acknowledgement. Finally, the pull-request was merged into the *Ruby on Rails*' master branch.

As the example depicted above, apart from *javan* who is informed by *rafaelfranca*, other three reviewers join the discussion spontaneously. Because all the comments affect

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

the decision-making of that pull-request, if they do not catch that pull-request timely, the time of review process would be longer. Beside, except for @*mentioning*, project managers can use a label to assign a pull-request to someone. However, only 0.89% of pull-requests have been set that label in our dataset.

Thus, if appropriate reviewers are automatically recommended when a new pull-request is submitted, its review process would be greatly accelerated. It is worth mentioning that this novel application can be seamlessly integrated with the @*mention* tool to take its full advantage by @*mentioning* the potential reviewers automatically.

III. Machine Learning based Recommendation

We aim to recommend highly relevant reviewers for pull-requests to improve the efficiency of social coding. The representative existing work of automated bug triaging are the approaches of mining bug repositories based on Machine Learning (ML) [7]–[9]. All these approaches start from the *bug-text*, such as title, description and source code. In the context of ML, the bug triaging problem can be represented as text categorization (i.e., classification of text documents into a set of categories). The text documents are the *bug-text* and the categories into which bug reports are classified are the names of developers suitable to resolve the report. In this paper, we treat pull-requests as text documents, and then utilize the ML-based approach to predict top-k relevant reviewers.

A. Vector Space Model of Pull-Request

Each pull-request 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 vector space model to represent each pull-request as a weighted vector. Each element in the vector is a term, and the value stands for its importance for the pull-request. For a given word, the more times it appear in a pull-request, the more important it is for that pull-request. On the contrary, the more pull-requests it appears in, the less important it is for distinguishing these pull-requests. 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 pull-request p_r, and N_{p_r} and N_{P_R} are the total number of terms in p_r and pull-requests in the corpus respectively.

B. Training ML Classifiers

In general, a pull-request would be reviewed by several developers, so ML classifiers should provide more than one label for a pull-request testing instance. It means that they should be able to deal with the multi-label classification problem [14]. A ranked list of recommended candidates is generated from top-1 to top-k according to the probability distribution that the ML classifiers predicted on the labels using the one-against-all strategy. When the probability values are equal, we rank the developers in terms of the number of pull-requests' comments that they had submitted to the given project. In this paper, we choose SVM as our basis classifier because it has been proved to be a superior classifier in developers recommendation [7], [12].

IV. Social Network based Recommendation

ML-based recommendation focuses on the text of pull-requests. However, the preprocessed text of a pull-request mainly consists of the names or identifiers of code files, functions and variables, so the corpus of a project is not very large. When a reviewer has commented plenty of pull-requests, his label covers most of terms in the corpus. The ML classifiers would biasedly assign almost all incoming pull-requests to him. Thus, the workload of this reviewer continues to increase.

By contrast, the basic intuition of social network based recommendation is that the developers who share common interests with a contributor are the appropriate reviewers of his incoming pull-requests. For reviewer recommendation, the common interests among developers can be directly reflected by comment relations between contributors and reviewers in historical pull-requests. We propose a novel approach to construct *comment networks* by mining historical comment traces. Then, we predict highly relevant reviewers to incoming pull-requests based on *comment network* analysis.

A. Comment Network Construction

For each project, the corresponding *comment network* is constructed individually. In a given project, the structure of comment relations between contributors and reviewers is a many-to-many model. As shown in Figure 3, there are many contributors have submitted pull-requests to Project *P*. A developer can be a contributor submitting several pull-requests, and he could also be a reviewer in other contributors' pull-requests. A pull-request would be commented by several reviewers more than once. For example, reviewer R_1 had presented 5 comments in the pull-request PR_2. In addition, a reviewer would inspect multiple pull-requests, such as reviewer R_1 has commented PR_1 and PR_2.

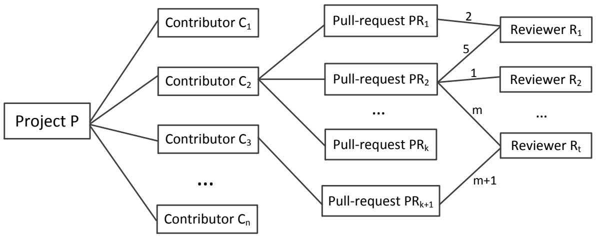


Figure 3. Comment relations between contributors & reviewers

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 pull-requests, there is a edge e_{ij} from v_i to v_j. The set of weights

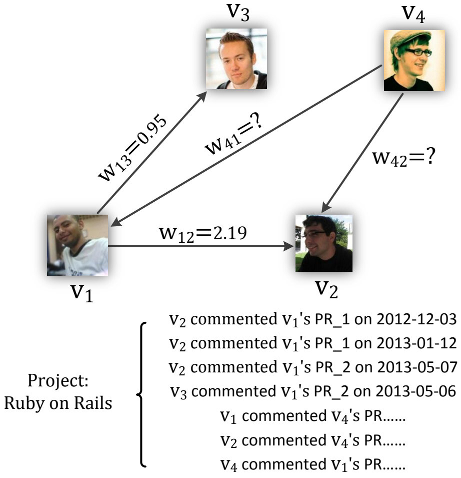
W reflects the importance degree of edges, and the weight w_{ij} of e_{ij} can be evaluated by Equation 2.

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

where k is the total number of pull-request submitted by v_i, and $w_{(ij,r)}$ is a component weight related to an individual pull-request 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 pull-request, and m is the sum of comments submitted by v_j in the same pull-request r. When reviewer v_j published multiple comments ($m \neq 1$) in the same pull-request, 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:

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

where $timestamp_{(ij,r,n)}$ is the date that reviewer v_j presented the comment n in pull-request 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 of the last one and a half years from 2012-01-01 to 2013-05-31 to learn the weights of *comment network*, the parameters $baseline$ and $deadline$ are set to 2011-12-31 and 2013-05-31 respectively.



v2 commented v1's PR_1 on 2012-12-03
v2 commented v1's PR_1 on 2013-01-12
v2 commented v1's PR_2 on 2013-05-07
v3 commented v1's PR_2 on 2013-05-06
v1 commented v4's PR......
v2 commented v4's PR......
v4 commented v1's PR......

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 pull-requests (`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 pull-request, 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 to v_3, which has been quantified by the corresponding weights of edges.

The *comment network* has several desirable qualities.

- Firstly, the global collaboration structure is revealed between contributors and reviewers in a given project, which can be used for mining reviewer candidates of incoming pull-requests.
- Secondly, 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.
- Thirdly, the decay factor λ is introduced to guarantee the difference values between the comments submitted to multiple pull-requests or single pull-request. For example, if reviewer v_j commented 5 different pull-requests of v_i and meanwhile v_q commented one of v_i's pull-requests 5 times, the weight of w_{ij} is larger than w_{iq}.

B. Reviewers Recommendation

Based on the *comment networks*, new pull-requests are divided into two parts according to their submitters. The first part are the *Pull-requests from Acquaintance Contributors* denoted as *PAC*. For a *PAC*, starting from the node of its submitter, we can find at least one neighbor in the directed graph. For example, in Figure 4, when v_1 submits a new pull-request, this pull-request is a *PAC* because we find two neighbors starting from v_1. The other part are *Pull-Requests from New Contributors* denoted as *PNC*. For a *PNC*, the submitter used to be a reviewer but has not submitted any pull-request, or it is a newcomer excluded from the training set, so there is no neighbor starting from it in the corresponding *comment network*. Hence, we can further divide reviewer recommendation into two different tasks.

Algorithm 1 Top-k recommendation for *PAC*

```
Require: $G_{cn}$ is the comment network of a given project; $v_s$ is the contributor of a new pull-request; $top_k$ is the number of reviewers of requirement;
Ensure: $recSet$ is a set of sorted reviewers;
$Q.enqueue(v_s)$ and $recSet \leftarrow \emptyset$
repeat
    $v \leftarrow Q.dequeue$ and $G_{cn}.RankEdges(v)$
    repeat
        if $top_k = 0$ then
            return $recSet$
        end if
        $v_{nb} \leftarrow G_{cn}.BestNeighbor(v)$
        $Q.enqueue(v_{nb})$ and $G_{cn}.mark(v_{nb})$
        $recSet \cup \{v_{nb}\}$ and $top_k = top_k - 1$
    until $G_{cn}.Neighors(v)$ all marked
until $Q$ is empty
return $recSet$
```

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

Recommendation for *PAC*: For a *PAC*, it is natural to recommend the user who has previously interacted with the contributor directly, i.e., the node that is a connected neighbor starting from the contributor node in the *comment network*. If there are more than one neighbor, the node with the highest weights get selected first. Hence, reviewer recommendation can be treated as a kind of directed graph traversal problem. In this paper, we improve the classical method of *Breadth-First Search* to recommend top-k reviewers for new pull-requests as shown in **Algorithm 1**. First of all, we initialize a queue and put the source node v_s onto this queue. Then, starting from the unvisited edge with the highest weight (*RankNeighors*) every time, we loop to select (*BestNeighbor*) and marked the nearest neighbor as a candidate. If the number of contributor's neighbors is less than top-k, we further visit the child nodes until top-k nodes are selected out.

Recommendation for *PNC*: For a *PNC*, since there is no prior knowledge of which developers used to review the submitter's pull-request, we want to predict the candidates who share common interests with this contributor by analyzing the overall structure of *comment network*.

Firstly, for a contributor who is a node but without any connected neighbor in the *comment network*, we mine the reviewers based on patterns of co-occurrence across pull-requests. For example, if v_2 and v_3 have reviewed plenty of pull-requests together, we can assume that they would share more common interests than others. Thus, when v_3 submitted a new pull-request (*PNC*), we recommend v_2 to review his pull-request, and vice versa. We use *Apriori* algorithm of association rule mining [15] to generate top-k frequent itemsets, and then rank the candidates according to their *supports*.

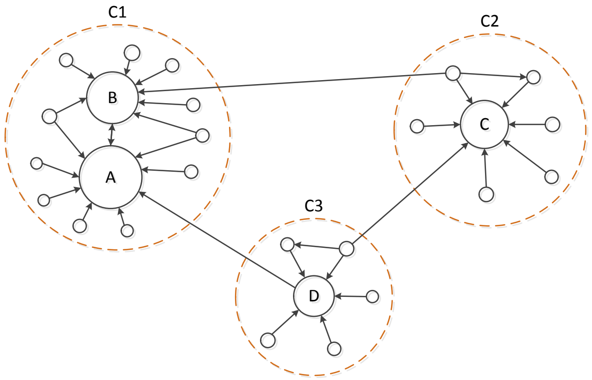


Figure 5. Community structure of *comment network*

In addition, for a newcomer who is a node excluded from the *comment network*, the most active reviewers in different kinds of communities become the most probable candidates. We assume that developers with common interests will spontaneously form a community in the *comment network*. However, the structure of the *comment network* shows that developers are not always uniformly dispersed, as shown in figure 5. It is probable that these most active reviewers could belong to the biggest community, such as the top-2 most active nodes A and B belong to the same community $C1$. Therefore, we would like our recommendation list to cover multiple communities, instead of always recommending from the biggest communities. In our implementation, we extract the community structure from the *comment network* using Gephi [16] which integrates and optimizes a famous algorithm of community detection purposed by Blondel et al. [17]. The size of the communities is depicted by the number of developers it has, and the activeness of a reviewer is measured by the number of pull-requests he has reviewed in history, in other words, the in-degree of a node in the *comment network*. The recommendation set is generated by following steps:

1) Rank the communities by their size (number of nodes);

2) calculate the in-degree of nodes in top-k community;

3) Select the top-k nodes based on their in-degree from top-k communities one by one.

V. Experiments Evaluation

A. Experiment Questions

RQ1: How effective is the SVM-based approach applied to pull-request assignment?

RQ2: Does the *CN*-based approach have significant improvement compared to the SVM-based approach?

RQ3: Do SVM-based approach and *CN*-based approach have consistent performance on projects of different characteristics?

For experiment question RQ1, we explore whether highly relevant reviewers can be identified using the machine learning approach described in section III. For experiment RQ2, we illustrate whether social relations are more effective for reviewer recommendation. For RQ3, we aim to know are there any distinct results of applying these approaches on different projects and why.

Table I
Preprocessed experiment dataset

Project	Language	#Pull-requests		#Comments		#Candidates
		Training	Test	Training	Test	
akka	Scala	1112	34	10640	467	16
scala	Scala	2028	64	13475	944	32
bitcoin	TypeScript	1067	63	5446	688	21
node	JavaScript	1196	14	3497	220	73
jquery	JavaScript	512	14	2967	196	27
symfony	PHP	2029	29	9601	419	79
phantomjs	C++	677	53	2866	728	20
xbmc	C	1629	81	8493	1094	73
rails	Ruby	3158	59	13060	583	168
homebrew	Ruby	4307	51	12255	842	22
Sum		17715	462	82300	6181	479

B. Experiment Design

Data Selection: Gousios et al. [18] [13] provide a comprehensive dataset to study the pull-request development model of GitHub. Our approach is evaluated on 10 projects which have received over 1000 pull-requests. We use the data of last one and a half years from 2012-01-01 to 2013-05-31 as our training set and the data from 2013-06-01 to 2013-10-01 as our test set. The descriptions and titles are used to learn SVM classifiers and comment relations are used to build *comment networks*. In order to learn the valid classifiers, we first do stop words removal and stemming over the descriptions and titles. Then, we retain those pull-request with more than 10 words. The test set includes some simple pull-requests which need not be reviewed, so this kind of pull-requests which received less than 4 comments are omitted. After that, there are 17715 pull-requests in the training set and 462 in the test set, as shown in Table I. For each project, we recommend top-1 to top-10 reviewers to a new pull-request from *Candidates* in Table I.

Evaluation Metrics: We evaluate the performances of our approaches over each project by precision and recall which are widely used as standard metrics in previous work. The formulae for our metrics are listed below:

$$Precision = \frac{\mid Rec\_Reviewers \cap Actual\_Reviewers \mid}{\mid Rec\_Reviewers \mid}$$
$$Recall = \frac{\mid Rec\_Reviewers \cap Actual\_Reviewers \mid}{\mid Actual\_Reviewers \mid} \quad (4)$$

C. Experiment Results and Analysis

Baseline: Most Active Recommendation

It is common for some projects that most of pull-requests are reviewed by a few core developers. Thus, to demonstrate the effectiveness of our approaches based on machine learning and social network analyzing, we design a baseline method that every new pull-request is assigned to the top-k most active developers ranked according to the number of pull-requests they have reviewed in the past.

RQ1 & RQ2: Recommendation Performances

We use a chart of *precision vs. recall* to show the performances of different approaches in detail. In Figure 6, each curve has a point for each recommendation from top-1 to top-10. There is a trade-off between precision and recall for classification. Hence, the precisions are gradual decreasing with the increase of recalls.

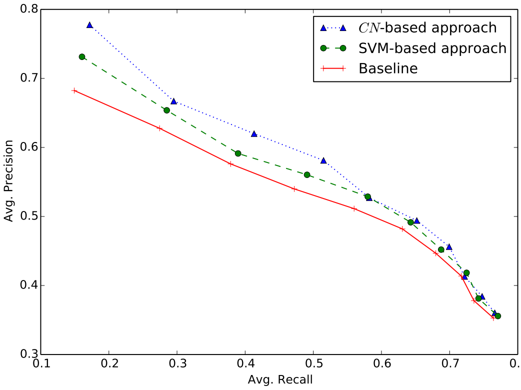


Figure 6. Precision vs. Recall of different approaches

Both SVM-based approach and *CN*-based approach are superior to the baseline. Our novel approach based on *Comment Network* (*CN*-based approach) achieves obvious improvement in precision from top-1 to top-4, especially at the point of top-1 (78%) and top-2 (67%). However, the curve of SVM-based approach is in the middle from top-1 to top 4, and takes the leading position at the points of top-5 and top-8 in precision. When we recommend top-10 reviewers, the recall of each method can reach the level of 77%.

Furthermore, we present the following null hypotheses to evaluate the improvement of our *CN*-based approach compared with the baseline and the SVM-based approach:

H_{0_1}: *There is no statistically significant difference between the precisions of the baseline and CN-based approach.*

H_{0_2}: *There is no statistically significant difference between the precisions of the SVM-based and CN-based approach.*

H_{0_3}: *There is no statistically significant difference between the recalls of the baseline and CN-based approach.*

H_{0_4}: *There is no statistically significant difference between the recalls of the SVM-based and CN-based approach.*

The results of Student's t-test are listed in Table II. For $\mathbf{H}_{0\_1}$ and $\mathbf{H}_{0\_2}$ of precision, one-sided p-value $p < 0.05$ and t-test value $t < t_{crit}$. Thus, we reject these hypotheses. It means that *CN*-based approach achieve a significant improvement in precision compared to the baseline and the SVM-based approach. Similarly, we reject $\mathbf{H}_{0\_3}$ and $\mathbf{H}_{0\_4}$ of recall, but the t-test result of $\mathbf{H}_{0\_4}$ is remarkably close to the critical values. It means *CN*-based approach achieves nearly the same performance in recall but greatly exceeds the baseline method.

Especially, high precision of top-k recommendation is significant for pull-request assignment. As the example of Figure 1, if our approach assigns that pull-request to *rafaelfranca*, he would remind *javan* to join the discussion by the *@mention* tool, even if *javan* is left out by the algorithm.

Table II
RESULTS OF T-TEST FOR OUR HYPOTHESES

H	Var	Approach	μ	σ^2	p	t	t_{crit}	Decision
H_{0_1}	Precision	CN-based baseline	0.53 0.50	0.031 0.029	0.0002	3.67	1.66	Reject
H_{0_2}	Precision	CN-based SVM-based	0.53 0.51	0.031 0.030	0.016	2.18	1.66	Reject
H_{0_3}	Recall	CN-based baseline	0.56 0.54	0.057 0.063	0.0003	3.60	1.66	Reject
H_{0_4}	Recall	CN-based SVM-based	0.56 0.55	0.057 0.063	0.040	1.77	1.66	Reject

Means μ, Variance σ^2, statistical significance p-value p, t-test results t and critical value t_{crit}, significance level $\alpha = 0.05$. In this paper, we do not list alternative hypotheses, because they are easy to derive from these null hypotheses respectively.

RQ3: Discussion of different approaches

We run our approaches on each project to evaluate their detailed performances and discuss the results. As most of pull-request in GitHub received less than 4 comments [1], we only depict part of our results from top-1 recommendation to top-5.

Firstly, running on *bitcoin*, *akka* and *rails*, our novel *CN*-based approach can achieve a significant improvement over the baseline and SVM-based method. As shown in Table III, the improvement of *CN*-based approach running on each project is on average over 10% in precision and 6% in recall. In addition, *CN*-based approach can get remarkable performance for top-1 recommendation. For example, the precision of *CN*-based approach can reach 92% running on *bitcoin*. Compared to the baseline and SVM-based method, our approach achieves great improvement of 51% and 31% respectively.

In these three projects, the social activities and contributions between core and external developers are balanced and well-distributed. For example, for *bitcoin*, the number of core developers and external developers are equal in the history of top-10 active reviewers list. In addition, 64% of pull-requests

Table III
PRECISIONS/RECALLS OF BASELINE (BL), *CN*-BASED AND SVM-BASED RECOMMENDATION FROM TOP-1 TO TOP-5

Number of	bitcoin			akka			rails		
Reviewer	BL	CN	SVM	BL	CN	SVM	BL	CN	SVM
top-1	0.41/0.08	**0.92/0.19**	0.61/0.12	0.61/0.14	**0.84/0.20**	0.87/0.20	0.61/0.13	**0.75/0.18**	0.62/0.13
top-2	0.69/0.28	**0.77/0.31**	0.69/0.28	0.69/0.32	**0.84/0.39**	0.69/0.32	0.46/0.20	**0.65/0.32**	0.45/0.20
top-3	0.69/0.42	**0.73/0.44**	0.69/0.42	0.61/0.43	**0.70/0.50**	0.69/0.49	0.47/0.31	**0.57/0.42**	0.46/0.30
top-4	0.61/0.48	**0.67/0.54**	0.65/0.51	0.67/0.63	**0.73/0.69**	0.75/0.70	0.40/0.35	**0.46/0.45**	0.41/0.36
top-5	0.60/0.58	**0.63/0.62**	0.61/0.59	0.72/0.84	**0.78/0.92**	0.80/0.93	0.35/0.38	**0.38/0.47**	0.36/0.39
Avg.	0.60/0.36	**0.74/0.42**	0.65/0.38	0.66/0.47	**0.78/0.54**	0.76/0.53	0.46/0.27	**0.56/0.36**	0.46/0.28
Number of	jquery			phantomjs			homebrew		
Reviewer	BL	CN	SVM	BL	CN	SVM	BL	CN	SVM
top-1	**0.82/0.17**	**0.82/0.17**	**0.82/0.17**	**0.92/0.20**	**0.92/0.19**	**0.94/0.20**	**0.91/0.21**	**0.88/0.20**	**0.88/0.21**
top-2	0.66/0.27	0.70/0.29	0.68/0.28	0.86/0.37	0.74/0.31	0.88/0.37	0.76/0.35	0.77/0.35	0.77/0.35
top-3	0.53/0.33	0.56/0.35	0.55/0.34	0.73/0.46	0.75/0.48	0.75/0.48	0.81/0.56	0.80/0.55	0.80/0.56
top-4	0.45/0.37	0.55/0.46	0.48/0.39	0.75/0.64	0.74/0.62	0.75/0.64	0.79/0.72	0.75/0.68	0.79/0.73
top-5	0.46/0.48	0.50/0.53	0.48/0.51	0.65/0.68	0.65/0.68	0.66/0.69	0.74/0.84	0.62/0.71	0.73/0.84
Avg.	0.59/0.32	0.63/0.36	0.60/0.34	0.78/0.47	0.7/0.46	0.80/0.48	0.80/0.54	0.76/0.50	0.80/0.54

submitted by external contributors and 60% of them have been merged. By contrast, 36% of pull-requests are originating from core developers but 84% of them have been merged.

The second group includes *jquery*, *phantomjs* and *homebrew*, where both our approaches and the baseline method can achieve high precision and recall. The the second part of Table III lists the results. For *phantomjs* and *homebrew*, the precisions and recalls of all methods on average reach approximately 80% and 50% respectively.

In these projects, a few developers seems to dominate the pull-request review activity. As they have submitted comments too frequently, it is easy to assign pull-requests of test set to them. Thus, the improvement of *CN*-based approach is not obvious. Taking *phantomjs* as an example, the most active user (ID *136322*) has reviewed 77% of pull-requests and interacted with 83% of contributors in training set. All algorithms tend to assign the new pull-requests to him first.

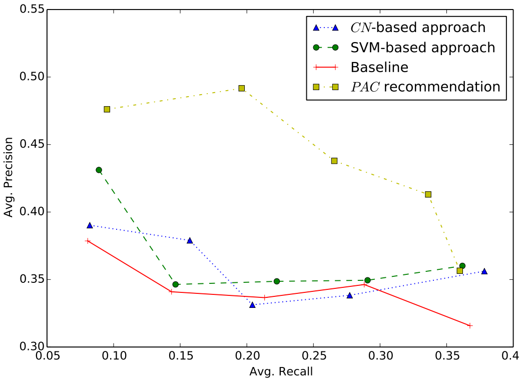


Figure 7. Precision vs. Recall of different methods for *xbmc*

However, all approaches do not perform well for *xbmc*. Except that SVM-based approach gets 43% precision for top-1 recommendation, the precisions of all methods ranged from 39% to 32%. In the test set of *xbmc*, 65% of pull-requests are originating from newcomers, so *CN*-based approach do much work of *PNC* recommendation (*PAC* and *PNC* have been defined in section IV-B). As shown in Figure 7, if we only consider to recommend reviewer for *PAC*, the performance of *CN*-based recommendation is greatly superior to others.

D. Conclusions of Experiments

Based on the experiment's results and analysis above, we can draw our conclusions as follows:

- For **RQ1**, the SVM-based recommendation is an effective approach for pull-request assignment which achieves 73% precision of top-1 recommendation and 77% recall of top-10 recommendation.
- For **RQ2**, the *CN*-based approach achieve significant improvement compared to other methods. Analyzing the social network is a novel and effective way for reviewer recommendation.
- For **RQ3**, the *CN*-based recommendation show much more improvement for the projects where the social activities are balanced between core developers and external developers. If developers have reviewed pull-requests actively, the performance of our approach is remarkable.

E. Threats to Validity

In this section, we discuss some threats to validity which may affect the experiment results of our approaches. Firstly, there is a small part of the core developers who are in charge of the final decision of pull-requests. They joined so many pull-requests' discussions in training set that all the approaches tend to assign the new pull-request to these active developers first. Hence, the workload of these active reviewers may not be reduced. However, with more and more external contributor presenting their suggestions to pull-requests, the social network based approach can refresh the weights of corresponding edges, so new pull-requests would be assigned more balanced than before. Besides, because some pull-requests have not been closed when we dumped the test set, so a part of following reviewers have not been taken into consider which may affect our recommendation results.

VI. RELATED WORK

Gousios et al. [1] show that the pull-request model offers fast turnaround, increased opportunities for community engagement and decreased time to incorporate contributions. To the best of our knowledge, this paper is the first one to study pull-request assignment. We review previous work about bug triaging and collaboration network analysis of social coding.

There are a number of researches based on Machine Learning (ML) [7]–[9] and Information Retrieval (IR) [10]–[12] techniques to triage incoming bug reports or change requests. Anvik et al. [7] apply a machine learning algorithm to learn the kinds of reports each developer resolves and recommend developers for a new bug report. Jeong et al. [8] find that many bugs have been reassigned to other developers, so they combined classifiers with tossing graphs to mine potential fixers. Based on that, Bhattacharya et al. [9] introduce an approach adding fine-grained incremental learning and multi-feature tossing graphs to reduce tossing path lengths and improve prediction accuracy. Other researchers use IR for automatic bug triaging. Canfora and Cerulo [10] use the textual descriptions to index developers and source files as documents in an information retrieval system. Kagdi [11] and Linares-Vasquez [12] extract the comments and identifiers from the source code and index these data by Latent Semantic Indexing. For a new bug report, such indexes can be useful to identify the most appropriate developers to resolve it.

In this paper, we not only expand the ML-based methods to recommend reviewers for incoming pull-request, but also propose a novel social network based approach focusing on mining social relations betweens reviewer and contributors which achieves a significant improvement.

The social coding paradigm is reshaping the distributed software development in recent years with a surprising speed and wide range. Dabbish et al. [4] explore the value of social mechanisms of GitHub. They find that the transparency in collaboration is improved by social mechanisms which is significant for innovation, knowledge sharing and community building. Zhou et al. [19] measure the behavior of a contributor using workflow network. Zanetti et al. [20] categorize bugs based on nine measures to quantify the social embeddedness of bug reporters in the collaboration network. Thung et al. [21] investigate the developer-developer and project-project networks of GitHub. They use PageRank to identify the most influential developers and projects in Github.

These previous researches inspire us to mining the *comment networks* for pull-request automatic assignment.

VII. Conclusion and Future Work

The review process of pull-request is important for social coding. In this paper, we firstly extend the machine learning based approach for bug triaging to assign reviewers to new pull-requests. Furthermore, introducing social relations, we propose a novel social network based approach for reviewer recommendation by mining a new type of social network called *comment network*. Finally, we demonstrate the effectiveness of these two approaches with quantitative evaluations. In the future, we plan to use the information retrieval approaches to pull-request assignment, and then combine these traditional methods with the social network based approach. Besides, we will explore how to add other types of social relations into our framework.

VIII. Acknowledgement

This research is supported by the National High Technology Research and Development Program of China (Grant No.2012AA011201), the National Science Foundation of China (Grant No.61432020 and No.61472430) and the Postgraduate Innovation Fund of University of Defense Technology (Grant No.B130607).

References

[1] G. Gousios, M. Pinzger, and A. van Deursen, "An exploration of the pull-based software development model," in *Proceedings of the 36th International Conference on Software Engineering*, Jun. 2014, to appear.

[2] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks," in *Proceedings of the 2006 International Workshop on Mining Software Repositories*, 2006, pp. 137–143.

[3] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and mozilla," *ACM Trans. Softw. Eng. Methodol.*, vol. 11, no. 3, pp. 309–346, Jul. 2002.

[4] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in github: transparency and collaboration in an open software repository," in *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*. New York, NY, USA: ACM, 2012, pp. 1277–1286.

[5] T. LaToza, W. Ben Towne, A. van der Hoek, and J. Herbsleb, "Crowd development," in *Cooperative and Human Aspects of Software Engineering, 2013 6th International Workshop on*, May 2013, pp. 85–88.

[6] 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.

[7] 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.

[8] G. Jeong, S. Kim, and T. Zimmermann, "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*, ser. ESEC/FSE '09. New York, NY, USA: ACM, 2009, pp. 111–120.

[9] P. Bhattacharya, I. Neamtiu, and C. R. Shelton, "Automated, highly-accurate, bug assignment using machine learning and tossing graphs," *Journal of Systems and Software*, vol. 85, no. 10, pp. 2275–2292, 2012.

[10] G. Canfora and L. Cerulo, "Supporting change request assignment in open source development," in *Proceedings of the 2006 ACM Symposium on Applied Computing*, ser. SAC '06. New York, NY, USA: ACM, 2006, pp. 1767–1772.

[11] 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.

[12] 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.

[13] G. Gousios, "The ghtorrent dataset and tool suite," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 233–236.

[14] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview." *IJDWM*, vol. 3, no. 3, pp. 1–13, 2007.

[15] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proceedings of the 20th International Conference on Very Large Data Bases*, ser. VLDB '94. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994, pp. 487–499.

[16] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: an open source software for exploring and manipulating networks." in *ICWSM*, 2009.

[17] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, 2008.

[18] G. Gousios and A. Zaidman, "A dataset for pull request research," in *MSR '14: Proceedings of the 11th Working Conference on Mining Software Repositories*, may 2014, to appear.

[19] M. Zhou and A. Mockus, "What make long term contributors: Willingness and opportunity in oss community," in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 518–528.

[20] M. S. Zanetti, I. Scholtes, C. J. Tessone, and F. Schweitzer, "Categorizing bugs with social networks: A case study on four open source software communities," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 1032–1041.

[21] F. Thung, T. F. Bissyande, D. Lo, and L. Jiang, "Network structure of social coding in github," in *Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering*, ser. CSMR '13. Washington, DC, USA: IEEE Press, 2013, pp. 323–326.