A Method for Recommending Bug Fixer Using Community Q&A Information

Qingjie Wei[1], *Jiao* Liu[2, *], and *Jun* Chen[2]

[1]Computer Science and Technology School, Chongqing University of Posts and Telecommunications, China
[2]Software Engineering School, Chongqing University of Posts and Telecommunications, China

Abstract. It is a very time-consuming task to assign a bug report to the most suitable fixer in large open source software projects. Therefore, it is very necessary to propose an effective recommendation method for bug fixer. Most research in this area translate it into a text classification problem and use machine learning or information retrieval methods to recommend the bug fixer. These methods are complex and over-dependent on the fixers' prior bug-fixing activities. In this paper, we propose a more effective bug fixer recommendation method which uses the community Q & A platforms (such as Stack Overflow) to measure the fixers' expertise and uses the fixed bugs to measure the time-aware of fixers' fixed work. The experimental results show that the proposed method is more accurate than most of current restoration methods.

1 Introduction

In the process of software development, bugs are unavoidable and it is important to fix bugs in time. To guarantee the quality of software, many projects use bug reports to collect and record bugs. Then, Bug triager assign the bug to a list of developers that are qualified to understand and fix the bug report, and then ranking them according to their expertise[1]. However, with the rapid development of software (i.e., especially open source software), vast amounts of bug reports have been produced. According to statistics, from October 2001 to December 2017, the number of bug reports for Eclipse accumulated about 530 thousand, with an average of about 100 bug reports per day. If these bug reports are handled manually only by a bug triager, it is very time-consuming and tedious task, which consequently impact the efficiency of bug fixing and thus increase the cost of the entire project.

To solve the above problems, many researchers proposed various methods to recommend bug fixer. Most of the methods related to machine learning, which can be divided into three parts: characterizing bug reports, training classifier and adjusting recommend result. Different researchers have carried out different studies for different parts. In the first part, Baysal et al. [2], Zhang T et al. [3] applied Vector Space Mode(VSM), Latent Dirichlet Allocation(LDA) respectively to character bug reports in the feature space to achieve the goal of reducing the dimension. In the second part, Fabrizio et al. [4], Anvik J [5], Zhang W et al. [6] used Naïve Bayes, Support Vector Machine(SVM), k-Nearest Neighbor(kNN) respectively to train classifier. In the third part, Jonsson et al. [7] choose the k developers with the highest probability to build a potential fixer recommendation list. Jeong et al. [8], Bhattacharya et al. [9] built a bug tossing graph using fixed bugs' tossing information, then, updated the recommendation list using the tossing graph to recommend a better potential fixer. However, these methods have an obsolete training set problem and the fixer recommended must fixed some bug reports similar to the newly one.

Information retrieval also applied to recommend fixer. Matter et al. [10] proposed a fixer recommendation method based on the contribution rate which collected from the developers' source code commits and the bug report keywords. Alenezi et al. [11] investigated the use of five term-selection techniques to select the most different terms in bug reports. The aim is reduced dimension and saved the time of recommendation. However, these methods can not recommend a new developer.

Considering that many developers access and contribute information to some open community question-answering platform(CQA), such as Java Forum, Yahoo! and Stack Overflow. These platforms have collected a lot of information that reflects the professional capabilities of developers. Sajedi et al. [12] proposed a method named $RA\_SSA\_Z\_score_{u,b}$ using question-answering (Q&A) information in Stack Overflow platform as evidence for the fixers' expertise to recommend fixer. Based on the research of Sajedi et al., this paper proposes a more optimized and efficient fixer recommendation method, which further studies how to use the information of CQA platform to measure the expertise of developers and use the fixed bugs to measure the timeliness of candidate fixers' fixed work.

* Corresponding author: 804830918@qq.com

2 Related concepts

2.1. Bug report

Bug report is a software document describing software bugs and helping developers locate and fix bug quickly, so as to ensure the quality of software projects. The main components of a typical bug report include Bug Id, its title, the resolution status (e.g., open, resolved) of the bug, when it is reported and modified, its reporter, its fixer, the description of the report [13].

2.2 Stack overflow

Stack Overflow is a popular technology related IT technology Q&A platform. On this platform, users can submit question, browse questions, and index related contents free of charge. Fig 1 shows a question information from Stack Overflow. We can see that, for any question, Stack Overflow uses some specific tags to divide the domain of the question, which can help users to locate the question quickly. Each question information has the number of answers, upvotes, and user information who answered this question. These Q&A information (also called posts), tags, upvotes are a great potential to measure the expertise of the developer.

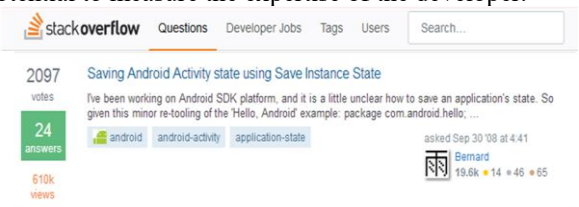

Fig 1. An example of question Info. from Stack Overflow.

3 Fixer recommendation method

Compared with traditional software development, software development today is more like a community activity. Developers often work on some projects hosted on large-scale software repository platform [14], which support code sharing, such as GitHub. Besides, developers also have technical exchanges on a number of community Q&A platforms when they encounter technical problems [15], such as Stack Overflow. Therefore, these two platforms have gathered a lot of information that can reflect the expertise of the developer [16]. Considering the cross-information of common users who are active on these two platforms, this paper will introduce in detail how to make use of Q&A information on Stack Overflow more reasonably and efficiently and recommend bug fixers to GitHub's bug reports. The framework is shown as Fig 2. From Fig 2, the core problem that needs to be solved in the whole method is how to calculate the expertise score and time-aware score of candidate fixers.

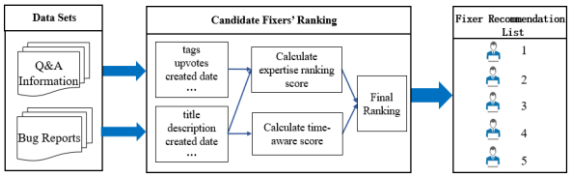

Fig 2. Framework of fixer recommendation.

3.1. A method of calculate expertise score

For developers' answer information in Stack Overflow is often the case that he or she knows this field well. On the contrary, developers' question information in Stack Overflow is the evidence of lack of relevant expertise. Based on this, we surmise that a developer makes $n = a + q$ posts, q of them questions and a of them answers. In addition, in order to eliminate the influence brought by different activity of different developer, this paper takes the ratio of $a - q$ and n as a measure of expertise, and get QA_score:

$$QA\_score = \frac{a-q}{n} = \frac{a-q}{a+q} \tag{1}$$

Given a bug report, QA_score uses the number of posts to reflect developers' expertise to a certain degree but ignores the correlation between the posts and bug reports. This paper uses Stack Overflow tags to cross-referencing bug reports in GitHub and posts in Stack Overflow. The tags of the Stack Overflow are a number of technical tags, such as Java, JavaScript, etc. These tags represent the main content of question information and its main function is to divide the question information into well-defined categories to facilitate users to search for areas of interest. Using Stack Overflow tags to match GitHub's bug reports text information, it is equivalent to providing developers with a common language to exchange information without needing to remove text information, stop words and other processing.

In addition, different tags have different relevance with bug reports, so the importance of tags in the text information of bug reports is different. Therefore, these tags cannot be measured with a unified standard. This paper sets different weights for different tags. Furthermore, the size of the number of upvotes reflects the quality of the posts. The more upvotes this post collects, the more expertise this post is assumed to be. Based on this, this paper redefines a and q as A_score and Q_score:

$$A\_score = \sum_{\substack{i \in tags\ in\ answers \\ posted\ by\ fixer}} (upV_a + 1) * w_{ai,b} \tag{2}$$

$$Q\_score = \sum_{\substack{i \in tags\ of\ questions \\ posted\ by\ fixer}} \frac{w_{qi,b}}{(upV_q + 1)} \tag{3}$$

A_score represents the score of all the answer information associated with the bug report on the Stack Overflow by the candidate fixer. upV_a represents the number of upvotes obtained from each answer

information (plus one for this answer itself). i represents the number of tag of this answer and $w_{ai,b}$ represents the weight of tag i in the bug report.

Q_score represents the score of all the question information associated with the bug report on the Stack Overflow by the candidate fixer. upV_q represents the number of upvotes obtained from each question information (plus one for this question itself). i represents the number of tag of this question and $w_{qi,b}$ represents the weight of tag i in the bug report.

Using A_score and Q_score to replace a and b respectively in formula (1), we get the way to measure the expertise of candidate fixer as ER_score (Expertise Ranking Score):

$$ER\_score = \frac{A\_score - Q\_score}{A\_score + Q\_score} \tag{4}$$

Note that, the created date of Q&A information used to measure expertise must be earlier than the date to build a bug report to be fixed, otherwise it has no meaning.

3.2. A method of calculate time-aware score

Different software projects require different expertise of developers. As developers work on different projects, their expertise shifts over time [17]. Motivated by the intuition that "more recent evidence of expertise of a developer is more relevant", this paper uses the historical fixed work of developers to measure the timeliness of work. We define it as TA_score (Time Aware Score):

$$TA\_score = \begin{cases} \sum_{j \in bugs\ of\ fixer} \frac{1}{(1+\Delta d(t_b, t_j))} \\ 0 \end{cases} \tag{5}$$

t_b represents the created date of the bug report to be fixed. t_j represents the created date of the fixed bug reports for a candidate fixer. $\Delta d(t_b, t_j)$ represents the difference in number of days between the j bug report fixed by the candidate fixer and the current bug report to be fixed. If the created date of these two bug reports, the value of $\Delta d(t_b, t_j)$ is 0. If a candidate fixer does not fix any bug reports before this current bug report to be fixed, the value of TA_score for this candidate fixer is 0.

3.3. Final ranking

The final bug fixer recommendation formula ER_TA_score is obtained by the formula (4) and formula (5). We define ER_TA_score as formula (6):

$$\begin{aligned} ER\_TA\_score = {} & \alpha * ER\_score \\ & + (1-\alpha) * TA\_score \end{aligned} \tag{6}$$

In this formula (6), α is the parameter that adjusts the weight of ER_score and TA_score. This paper sets the range of α in (0,1). By using ER_TA_score to calculate the matching scores of the candidate fixer, then we rank them according to the size of the scores from high to low. Finally, we select the top k ranked candidate fixer as the potential bug fixer.

4 Experiment

In this section, we introduce the experimental process and show the experimental results. Moreover, we compare the performance of our approaches and that of other previous studies.

4.1. Data Preparation

In order to demonstrate the effectiveness of the proposed approach, we choose the experimental data used by Sajedi et al.12. Table 1 shows the details of the two data sets.

Table 1. Data sets information.

Name	Data Sources	Time	Number
Bug Reports	GitHub	2010/06/22-2014/07/28	7144
Q&A Information	Stack Overflow	2008/07/31-2014/05/04	233801

These two data sets in Table 1 are selected from GitHub and Stack Overflow. The data set of bug reports comes from the top 20 projects (Table 2.) in GitHub with the highest number of community members and the highest number of bug assignees. All of the community members (CM) must have at least one Stack Overflow activity.

Table 2. The partial information of 20 projects.

Project Name	#Bugs	#CM	Project Name	#Bugs	#CM
rails	594	822	elasticsearch	548	79
angular.js	207	182	julia	255	73
docker	102	169	bundler	8	69
salt	736	167	khan-exercises	286	61
brackets	1274	123	scala	335	57
rust	454	122	yui3	318	54
fog	51	117	NServiceBus	335	51
travis-ci	91	102	edx-platform	93	40
gaia	449	97	www.html5rocks.com	142	37
framework	742	96	Ghost	124	28

4.2. Evaluation measure

This paper uses the average top-k accuracy and MAP (Mean Average Precision) to evaluate the effectiveness of the proposed method. The average top-k accuracy indicates that in all the reports to be fixed, the probability of a real fixer's ranking in the top k of the

recommended list. MAP is a kind of precise, synthesized, rank-based evaluation measure. The higher the overall rankings of recommended fixer are, the higher the MAP value is.

4.3. Results and analysis

The bug reports used in this experiment have been assigned to a member of the project, that is, every bug report corresponds to a real bug fixer. In the experiment, we put forward the method proposed in this paper for every bug report, calculate ER_TA_score and rank each project member to get a recommended list. Finally, we evaluate the effectiveness of this method according to the recommended results.

This experiment is divided into two parts. In the first part, we use three projects (*Julia*, *www.html5rocks.com* and *edx-platform* in Table 3) and a total of 490 bug reports to carry out the experiment. The purpose is to select the optimal tag weight function of formula (2) and formula (3), and adjust the parameters in formula (6). We set the number of tags matching each bug report and Q&A information to num, and set the same weight function to $w_{ai,b}$ and $w_{qi,b}$ as w. Table 3. shows the results of this experiment.

Table 3. The result of first experiment.

		top1(%)	top5(%)	top10(%)	MAP
ER_score	$1+\sqrt{num-1}$	18.87	64.82	86.09	0.390
	num	18.32	64.69	85.85	0.388
	$1+0.5*(num-1)$	**19.38**	**64.96**	**86.12**	**0.394**
	$1+(num-1)^2$	19.23	64.85	86.01	0.392
ER_TA_score	$\alpha=0.001$	44.90	87.76	92.45	0.632
	$\alpha=0.01$	**45.10**	**88.78**	**93.27**	**0.634**
	$\alpha=0.1$	45.31	85.92	90.82	0.629
	$\alpha=0.5$	45.42	86.94	92.24	0.632

From Table 3, this paper conducts experiments on the tag weights first. According to the experimental results, the accuracies of top1(%), top5(%), top10(%) and MAP for ER_score obtained the best results when $w=1+0.5*(num-1)$. Then, we adjusted the α in ER_TA_score. When α=0.01, the accuracy of top1(%) is not very high, but the accuracy of top5(%), top10(%) and MAP obtained the best results. Therefore, we finally set the weight of tag to $w=1+0.5*(num-1)$ and the parameter to α=0.01.

In order to further verify the effectiveness of the method proposed in this paper, we used the optimal parameters adjusted before and expanded the experimental data set (the rest of 17 projects in Table 2 with a total of 6654 bug reports). In addition, we compared the performance of our approaches and that of other previous studies, including 1NN, 3NN, 5NN, Naïve Bayes, SVM, RA_SSA_Z_score. The experimental results are shown in Table 4.

Table 4. The result of second experiment.

	top1(%)	top5(%)	top10(%)	MAP
1NN	43.09	70.46	86.27	0.575
3NN	46.48	75.63	86.51	0.610
5NN	45.60	75.00	84.41	0.596
Naïve Bayes	43.77	75.97	90.59	0.609
SVM	45.46	81.82	90.09	0.617
RA_SSA_Z_score	45.17	89.43	96.51	0.633
Our approach	**45.51**	**89.80**	**97.13**	**0.640**

From Table 4, we can see that the top k accuracies of our approach for k is 1, 5, 10 are 45.51%, 89.80% and 97.13% respectively. We also obtained the MAP as 0.640. Compared with other fixer recommendation methods, 3NN obtained the highest accuracy of top1(%) as 46.48%. However, for the more important accuracies of top5(%), top10(%) and MAP, the proposed method in this paper obtained the best results in Table 4.

Further analysis shows that the accuracy of top5(%) obtained 89.90%, which indicates that the most fixer recommended by our approach are within the top 5. If we recommend these 5 fixers to the bug triager to assign bug reports, compared to the original dozens, even hundreds of candidate fixers, greatly reducing the workload of bug triager and save a lot of time. Besides, compared with other fixer recommendation methods, the implementation process of our approach is much simpler and also does not need to train data or remove the stop words from bug report text information. These fully demonstrate the stability and effectiveness of our approach.

5 Conclusion

This paper presents an optimized method of recommending software bug fixer using community Q&A information, and fully excavates the cross-information between GitHub and Stack Overflow to achieve the recommendation of software bug fixers. First, our approach uses the Q&A information, tags and upvotes in Stack Overflow platform to measure the expertise of candidate fixers, and then uses the created date of fixed bug reports to measure the timeliness of fixing work; finally, combining with the two methods, we recommend bug fixers to fix a total of 7144 bug reports for 20 open source projects in GitHub. The experimental results show that the overall performance of the proposed method is superior to most of the recommended methods in the field, and has great application value.

References

1. J. Zhang, X. Y. Wang, et al. SCI China Inform. **58**(2):1-24 (2015).

2. O. Baysal, M.W. Godfrey, R. Cohen. *International Conference on Program Comprehension*. 297-298 (2009).
3. T. Zhang, G. Yang, B. Lee, et al. *Software Engineering Conference*. 223-230 (2015).
4. Fabrizio Sebastiani. ACM Comput. Surv..**34**(1):1-47 (2001).
5. J. Anvik. 2006. Who should fix this bug? *International Conference on Software Engineering*. 361-370 (2006).
6. W. Zhang, S. Wang, Q. Wang. Infor. Software Tech. **70**:68-84 (2016).
7. L. Jonsson, M. Borg, D. Broman, et al. Empir. Softw. Eng. **21**(4):1533-1578 (2016).
8. G. Jeong, S. Kim, T. Zimmermann. *Joint Meeting of the European Software Engineering Conference and the ACMSigsoft Syposium on the Foundations of Software Engineering*. 111-120 (2009).
9. P. Bhattacharya, I. Neamtiu. *IEEE International Conference on Software Maintenance*. 1-10 (2010).
10. D. Matter, A. Kuhn, O. Nierstrasz. *International Working Conference on Mining Software Repositories*. 131-140 (2009).
11. M. Alenezi, K. Magel, Banitaan S. J. Software. **8**(9):2185-2190 (2013).
12. A. S. Badashian, A. Hindle, E. Stroulia. *International Conference on Fundamental Approaches to Software Engineering*. 231-248 (2016).
13. T. Zhang, J. Chen, G. Yang, et al. J. Syst. Software. **117**(C):166-184 (2016).
14. A. S. Badashian, A. Esteki, A. Gholipour, et al. *International Conference on Computer Science and Software Engineering*. 19-33 (2014).
15. T. Komamizu, Y. Hayase, T. Amagasa, et al. *International Conference on Software Engineering and Knowledge Engineering*. 584-589 (2017).
16. B. Vasilescu, V. Filkov, A. Serebrenik. *International Conference on Social Computing*. 188-195 (2014).
17. R. Shokripour, J. Anvik, Z. M. Kasirun, et al. *Mining Software Repositories.* 2-11 (2013).