International Journal of Innovative Research in Advanced Engineering (IJIRAE) ISSN: 2349-2163
Volume 2 Issue 1 (January 2015) www.ijirae.com

Bug Triaging: Profile Oriented Developer Recommendation

Anjali Sandeep Kumar Singh
Department of Computer Science and Engineering,
Jaypee Institute of Information Technology

Abstract— Software bugs are inevitable and bug triaging is a difficult, time consuming, tedious and expensive task. For large software projects, the number of incoming bug reports is usually high. Triaging these large numbers of incoming bug reports is a difficult and time consuming task. Part of the bug triaging process is assigning a newly arrived bug report to a developer who could successfully resolve the bug. Assigning bug report to the relevant developer is an important step as it reduces the bug tossing. Bug tossing is the process of reassigning the bug report to another promising developer, if the first assignee is unable to resolve it. In this work, we propose a new approach for selecting the developers who have appropriate expertise in the related area for handling the bug reports. A profile is created for each developer based on his previous work. This profile is mapped to a domain mapping matrix which indicates the expertise of each developer in their corresponding area. In order to evaluate our approach, we have experimented with bug reports of chromium dataset. Our experimental evaluation shows that our proposed approach is able to achieve an efficiency of 86% for top-10 and 97% for top-20 developer ranking list.

Keywords— Bug Tossing, Domain Mapping Matrix, Bug Triaging, Activity Profile, Developer Recommendation

I. INTRODUCTION

Bugs are the programming errors that cause significant performance degradation. Bugs lead to poor user experience and low system throughput. Large open source software development projects such as Mozilla and Eclipse receive many bug reports. They usually use a bug tracking system where users can report their problems which occurred in their respective projects. Each incoming bug report needs to be triaged. For example, as of October 2012, the Mozilla project had received over 80,000 reports, averaging 300 new bug reports each day.

Selecting the most appropriate developer to fix a new bug report is one of the most important stages in the bug triaging process and it has a significant effect in decreasing the time taken for the bug fixing process [13] and the cost of the projects [7]. In traditional bug triage systems, a developer who is dominant in all parts of the project as well as the activities plays the role of bug triager in the project. The triager reads a new bug report, makes a decision about the bug, and then selects the most appropriate developer who can resolve the bug.

Fixing bug reports through the traditional bug triage system is very time consuming and also imposes additional cost on the project. For example, Eclipse has 239 active developers as on January 2011 and 282 modified files on the Eclipse platform project. Hence, many researches have been done to make the traditional bug assignment efficient and automatic.

One of the important reasons why bug triaging is such a lengthy process is the difficulty in selection of the most competent developer for the bug kind. The bug triager, the person who assigns the bug to a developer, must be aware of the activities (or interest areas) of all the developers in the project. Bug triaging normally takes 8 weeks to resolve a bug. If the developer, to whom the bug report is assigned, could not resolve it, it is assigned to another developer. This would consume both time and money. Thus, it is really important on part of bug triager to assign the bug report to a developer who could successfully fix the bug without need of any tossing. Hence, the job of bug triager is really crucial.

A number of approaches exist to recommend developers, e.g. using machine learning [8], location based approaches [3], social network based approaches [5] or Information extraction approaches [2]. These approaches train a classifier with different methodologies and use it to recommend developers for a new bug report. Paper [3] presented a two - phased location based approach for assignee recommendation based on the predicted location of the bug. Firstly, the source code files to be altered to fix the bug are predicted and then patches and commit messages associated with the source code files are used for linking the bug report to developers.

Paper [2] proposed a natural language processing (NLP) and Information Extraction (IE) based methodology to develop a recommendation system. Information needed to select the best suited developer is extracted from the version control repository of the project. It does not use the information of the bug repository. It assumed that the chances of developer fixing the bug are higher if the developer has previously worked on the source files that were changed to fix the bug.

Paper [1] presents an activity profile based approach for developer recommendation. An activity profile is created for each user from the previously fixed bug reports. This profile, to some extent, represents developer's level of expertise in a particular area. Our work is also related to their approach but we have used Domain Mapping Matrix[1] (DMM) to store the developer profiles. Thus, whenever a new bug report comes, we first find the bug tokens from the new bug report. Then recommend the developers who have maximum expertise in similar areas of the bug tokens using DMM.

We applied our approach to chromium bug repository and achieved an accuracy of 86% for top-10 and 97% for top-20 developer ranking list. This paper has made unique contributions by proposing a profile oriented developer recommendation approach for newly arrived bug reports. Based on the historical bug reports, the profiles of developers are created which indicates their area of expertise and their expertise factor. These obtained heuristics are then utilized to create a domain mapping matrix which is then used for ranking the developers.

The remainder of this paper is organized as follows. First, we summarized related work in section II. Section III illustrates the proposed method along with implementation details. Next, we evaluate the accuracy of the approach in section IV. Section V gives concluding remarks, followed by future directions for research in section VI.

II. RELATED WORK

Matter et al. [4] proposed an information retrieval based technique that computes the expertise of developers based on vocabulary used in source code. They extracted the vocabulary of developers from the version control repository of the project. Their approach finds a relationship between the vocabulary extracted from new bug reports and the vocabulary extracted from source code and then uses the relationship between them for recommendation purpose.

Wenjin et al. [12] proposed DREX (Developer recommendation with K- Nearest- Neighbour search and expertise ranking). Their approach utilizes the information of historical bug reports and its comments to find similar bug reports based on K- Nearest- Neighbours. Based on the frequency metric, ranking of developers is provided. Rather than finding similar bug reports in training dataset and then recommending the new bug report to those developers who resolved the similar bugs, our approach finds the developers who have maximum expertise in areas to which the bug report belongs. This approach searches for more proficient developers as the number of tokens matched is more in case of profiles. Thus our approach uses a new and wider concept of profile creation to efficiently recommend the developer.

Bhattacharya et al. [8] employed a fine grained incremental learning approach. They constructed multi feature tossing graphs and used it for predicting the relevant developers for the bug report. Their results showed high prediction accuracy for recommending developers and high reduction in tossing path lengths.

Hosseini et al. [9] proposed an auction based bug allocation mechanism. When a new bug report comes, the bug triager extracts some basic information such as type, severity, etc. and then auctions off the bug. The developers place their bids on the bugs. The bug triager receives the bids and based on the developer's experience, expertise, historical preferences and current work queue, he announces the final decision and assigns the bug report to the winner developer.

III. PROPOSED METHOD

Preliminaries:

Generally, a bug report consists of various predefined fields. In Chromium bug repository, fields such as "ID", "Bug Type", "Cr Value", "OS", "CC", "Summary", etc are present. For instances, "ID" refers to the unique sequence number provided to each reported bug. "Bug Type" refers to the type of bug report such as Bug Regression, Feature, etc. "Cr value" refers to the component to which bug belongs to. "OS" field refers to the operating systems in which the bug occurs such as android, iOS, Windows, etc. "CC" refers to the developers who are related with the bug report. The bug report contains multiple developer names in "CC" field. Because we do not know the actual contribution of each developer towards resolving that bug, we assumed that all developers in this field have equal expertise and contribution in resolving that particular bug report. "Summary" field constitutes short description about the bug report. Figure 1 shows the detailed information of the chromium bug report with ID 311655 (https://code.google.com/p/chromium/issues/detail?id=311655&can=7&colspec=ID%20Pri%20M%20Iteration%20ReleaseBlock%20Cr%20Status%20Owner%20Summary%20OS%20Modified%20Type&start=100).

[1] http://www.dsmweb.org/

Problem Statement:

Let us assume there are m bug reports B_1, B_2,…, B_m in chromium software project. These m bug reports are associated with various terms, T such that T (B_i) = {T_{i1}, T_{i2},…, T_{ik}} which are present in various predefined fields corresponding to bug reports. Each bug report Bi corresponds to a collection of developers, D (Bi) = {d_{i1}, d_{i2}, d_{i3},…, d_{il}}. The structure of a bug report in chromium bug repository is depicted in Figure 2. In this context, the objective of our work is to find most suitable developers for new bug report who could efficiently resolve the bug.

Research Goal:

The goal of this research is to use the term frequency mining technique to predict the most suitable developer for the new bug report depending upon the information available from historical bug reports. The terms (or topics) extracted from the previously fixed bug reports are used for creation of profiles of developers. The frequency of each term corresponding to each topic indicates the expertise factor of each developer relating to that particular topic.

A study of bug reports proved that there exists various similar terms in the bug reports and thus these terms can be used for linking the bug report to appropriate developer. We use this study to predict the most suitable developers for the newly arrived bug report.

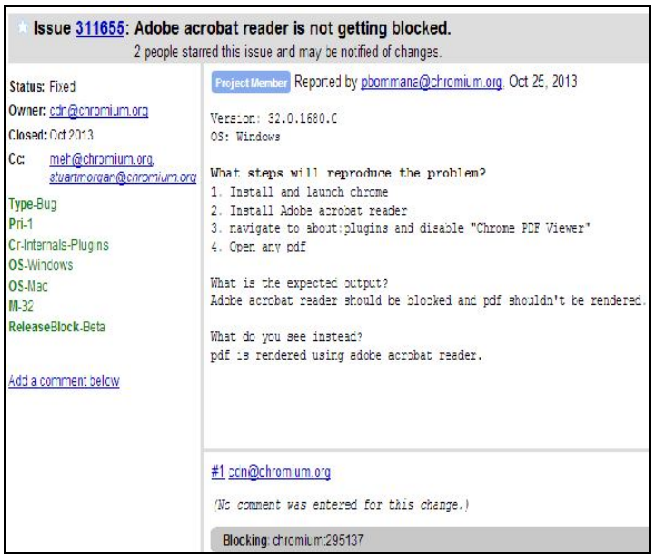

Fig. 1. The detailed information of the chromium bug report with ID 11655.

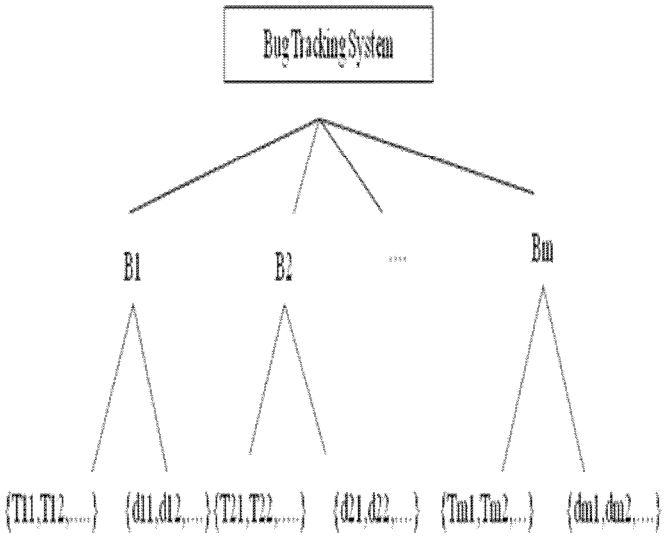

Fig. 2. The structure of chromium bug repository.

Approach:

In order to recommend a list of most suitable developers for fixing the bug report, we first generate tokens from the dataset (see Section III-A). Next, we obtain all the unique developers from the dataset. We link these developers to the bug reports of dataset they have worked upon. Using this heuristic, we link the developers to the tokens of bug reports. All the unique tokens corresponding to each unique developer are identified and the number of occurrences of each token gives the expertise factor of the developer related to that topic. These topics, expertise factors and developers constitute the DMM which forms the basis for recommendation (see Section III-B). When a new bug report comes, its terms are generated and then from the DMM the list of developers with highest aggregate expertise factor corresponding to all the terms of the new bug report are recommended (see Section III-C). None of the previous approaches have made use of DMM to find relevant developers.

A. Token Generation

In this step, we generate all the tokens from the dataset. The data we used for the project was obtained from the chromium bug tracking repository website[2]. The bug reports were downloaded in the form of CSV(Comma Separated Version) files. A single csv file contained 100 bug reports. We consider 1200 bug reports for developing profiles of the developers. Out of these 1200 bug reports, some reports contained no developer names or bug ids. We discarded such reports for training. The final collection contained 875 bug reports. We stored all these bug reports in the Oracle 10 g database. Oracle 10 g was chosen because it provides the best grid view for easy understanding.

We consider *bug id, cr- value, operating system, bug type* and *summary* field of these bug reports as these fields constitutes the most appropriate information relating to the bug reports. Selected fields of the bug reports are tokenized using the String Tokenizer. Tokens are filtered by removing stop words. The filtering process is done in an iterative manner. Porter Stemmer [15] tool is used to group similar words in the summary field together. For example, the terms “accept”, “acceptance”, “accepted”, “accepts” and “acceptable” are grouped together into a single term “accept”. This preprocessing reduces the noise and redundant data in the dataset and improves our automated triaging process.

Tokens obtained after the preprocessing are analyzed. The analysis showed that the tokens obtained from the *cr- value, operating system* and *bug type* are relevant and appears frequently in dataset whereas the list of tokens obtained from the *summary* field was so large that it would hamper the computations. Most summary tokens appear only once or twice in the complete dataset. Thus, their contribution in generating expertise factor is very less. This small frequency makes no difference in result. Table 1 represents the overview of summary tokens. Capobianco et al. [11] showed that using only unigram noun terms in the place of other parts of speech such as verbs or adjectives can reduce the computation and improves the efficiency. This is because when two people use the same noun term, their purposes are often related. But in our work, similar to tokens of summary, the frequency of noun tokens was so small that they made no difference in the result besides making computations difficult. Table 2 represents the overview of noun terms in summary tokens. Hence after final analysis, the tokens of summary field were discarded and hence they are not used for creating developers profile.

# Total Summary Tokens	4961
# Unique Summary Tokens	1497
# Tokens with frequency one	898
# Tokens with frequency two	418

Table 1: Overview of Summary Tokens

# Total Noun Tokens	3931
# Unique Noun Tokens	1276
# Noun Tokens with frequency one	847
# Noun Tokens with frequency two	170

Table 2: Overview of Noun Terms in Summary Tokens

B. Developer Profiling

After obtaining tokens from the bug reports, we create the profiles of developers. The chromium bug repository does not contain any list of all its developers. Thus to obtain developers from the dataset, we used *cc* field of each bug report. The *cc* field of bug reports contained the developers who are associated in resolving the bug. Thus, we obtained all the developer names from the bug reports. We found bug report ids corresponding to each developer i.e. all the bug report ids which the particular developer has contributed in resolving.

Bug Id	Tokens
1	a
2	a ,b
3	b ,c
4	a ,c

Table 3: Structure of Bug Report

Tokens	a	b	c
Karen	3	2	2

Table 4: Structure of Domain Mapping Matrix

Replace bug ids with the corresponding bug tokens to obtain a redundant list of tokens corresponding to each developer. Next we selected all the unique tokens corresponding to each developer from the obtained redundant list and calculated frequency of all unique tokens in the redundant list. These unique tokens reflect the expertise areas of the developer and the corresponding frequency of token reflects the expertise factor of developers in that particular area. Finally, we created a domain mapping matrix, M of size i*j, where i represents the total number of unique developers in the dataset and j represents the total number of uniquely identified tokens in the dataset.

$$M(i,j) = expertise\ factor\ of\ developer\ 'i'\ in\ area\ 'j' \quad (1)$$

The M (i, j) reflects the association of developer i with token j.

For example, if user *"Karen"* is associated with 4 bug reports with bug ids 1, 2, 3 and 4, which have certain associated tokens. Table 3 represents the structure of a bug report with its tokens. Then according to equation (1), profiles of developers in the form of Domain Mapping Matrix M will be generated as shown in Table 4. For each token 'a', 'b' and 'c', count their frequency from Table 3. The frequency of token 'a' is 3 (Bug id 1, 2 and 4), 'b' is 2 and 'c' is 2. Table 4 represents the partial domain mapping matrix for developer Karen. It shows that Karen had previously worked upon bugs associated with token 'a' thrice. So his expertise in token 'a' is 3. Similarly his expertise with other tokens is calculated. The final DMM (domain mapping matrix) represents the profiles of all developers in the dataset. This DMM is further used for developer recommendation.

C. Developer Recommendation

For each new bug report its tokens are generated (see Section III-A). Appropriate developer is found for the newly arrived bug report. For developer recommendation, consider each extracted token, t_j of the new bug report and sum up the expertise factor of each extracted term, t_j corresponding to each developer, i. For each developer i, expertise factor e_{ij} corresponding to tokens, t_j of newly arrived bug reports can be obtained by equation (2).

$$e_i = \sum_{j=0-n} M(i,j) \quad (2)$$

After obtaining the aggregate expertise of all the developers corresponding to all the tokens of new bug report, recommend a list of developers to the triager. This aggregate expertise is considered the most important element in generating the developer ranking list as it reflects the frequencies of activities performed by all the developers corresponding to all the tokens of new bug report. Thus, the developer with highest expertise factor becomes the most relevant developer. Sort the list in descending order. The obtained list gives the final ranking of developers for the new bug report.

Algorithm

1. Generate tokens of each bug report from three predefined fields i.e. '*cr- value*',' *operating system*' and '*bug type*'.
2. Construct a list of unique developers from '*cc*' field.
3. Corresponding to each developer, extract bug report ids on which developer has worked upon.
4. Replace the bug ids with corresponding bug report tokens.
5. Calculate all the unique tokens along with their frequency corresponding to each developer.
6. Construct the domain mapping matrix, M of size i * j, where i represents the total unique developers and j represents the total number of uniquely identified tokens. The M(i ,j) represents the expertise of developer 'i' in area 'j'.
7. Generate tokens of new bug report from three predefined fields i.e. '*cr- value*',' *operating system*' and '*bug type*'.
8. For each extracted token, t_j sum up the expertise factor e_{ij} corresponding to each developer.
9. Sort the list of expertise factors in descending order to obtain the list of most prominent developers who could successfully resolve the bug.

IV. EVALUATION

To evaluate our proposed work, we used 875 bug reports of the chromium bug repository for training and 65 reports for testing. Table 5 gives an overview of training and testing datasets. We found that these 875 bug reports were assigned to 682 unique developers in the dataset. After preprocessing of training dataset as mentioned in section III-A, the profiles of all the 682 unique developers are made by utilizing the information obtained from historical bug reports. These 682 developer's expertise is distributed over 152 unique tokens obtained from the training bug reports. To evaluate our approach, we have generated the tokens of testing bug reports. Corresponding to the tokens of these new bug reports, we used our approach to obtain ranked list of relevant developers.

TRAINING SET	
#REPORTS	875
#DEVELOPERS	682
TESTING SET	
#REPORTS	65

Table 5: Overview on Training and Testing Dataset

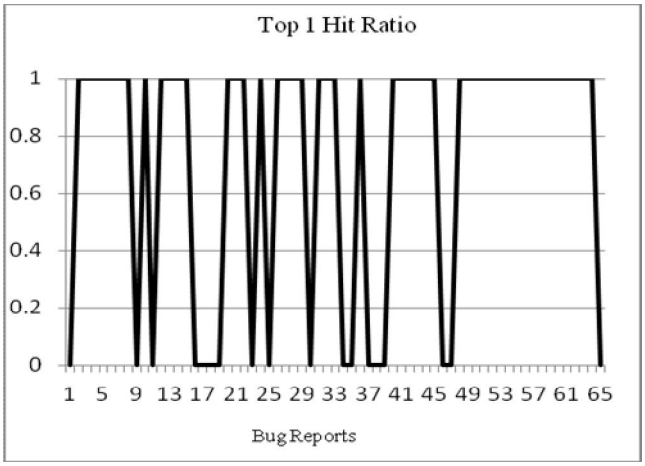

Fig. 3. Hit Ratio for Top 1 Ranking List

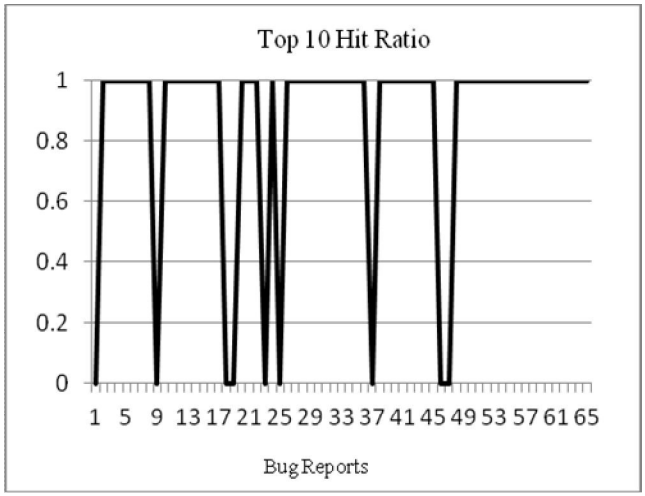

Fig. 4. Hit Ratio for Top 10 Ranking List

The next step is to evaluate whether the recommendation list contains useful developers. For this, we have made comparison by finding if the ranked list contains the actual developer who had been assigned the report earlier. In chromium bug repository, the '*cc*' field contains multiple developer names. Because we do not know the actual contribution of each developer towards resolving that bug, we assumed that all developers in this field have equal expertise and contribution in resolving that particular bug report. Thus, for comparison a hit is considered, if the recommended list contains any one of the developer who had been assigned the bug report. We evaluated the ranking by measuring top-*n* ranked lists. The hit percentage in top1, top10, top20 and top 30 ranked lists are computed for testing bug reports. Figure 3, 4, 5and 6 shows the hit ratio for top-1, top-10, top-20 and top-30 developers respectively. Our approach achieves a hit ratio of 73%, 86%, 97% and 97% for top-1, top-10, top-20 and top-30 developers respectively. Figure 7 shows the overall hit ratio obtained by profile based technique.

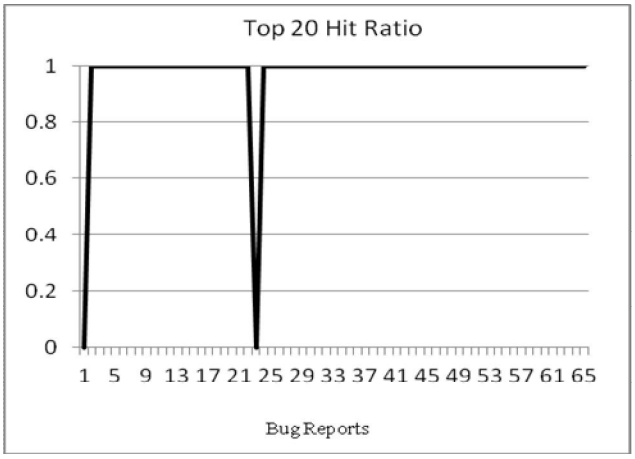

Fig. 5. Hit Ratio for Top 20 Ranking List

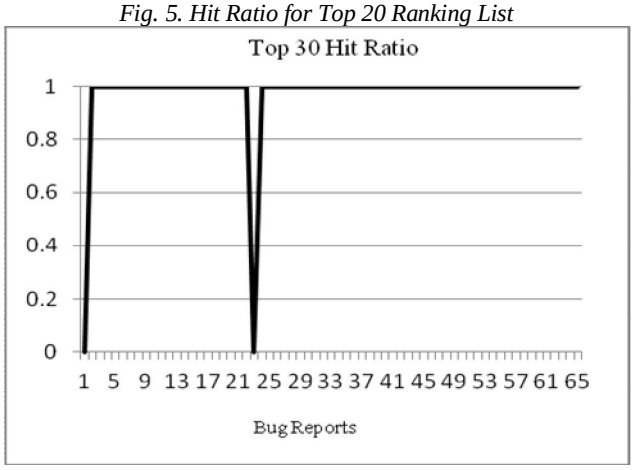

Fig. 6. Hit Ratio for Top 30 Ranking List

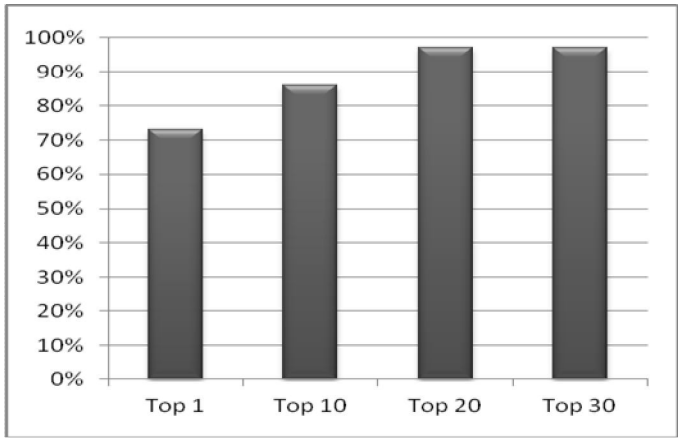

Fig. 7. Overall Efficiency of proposed approach.

V. CONCLUSION

In this paper, we proposed a Domain Mapping Matrix (DMM) based developer recommendation approach for predicting the best suited developers list who could resolve the newly reported bugs. Unlike other approaches, our approach does not find a matching with historical bug reports and recommend the developers who fixed historical bug report; rather, it utilizes the expertise profile of developers maintained in DMM. This profile can be easily updated with time. Using the developer profile is better as the number of token matching is more. For any new bug report all of its tokens are matched in the dataset and expertise list corresponding to new bug report tokens is generated. Thus, our proposed approach uses a wider area for token matching. We evaluated our approach by finding, if the recommended list contains the actual assignee of the report. Our approach uses text frequency mining approach to link the developers with appropriate tokens or expertise areas. The main goal of this research is to recommend the most suitable developer list for new bug reports and we have obtained hit percentage of 82% and 96% for top 10 and top 20 ranking lists.

VI. FUTURE WORK

In the future, we will apply our proposed approach to more bug repositories (e.g. Bugzilla) for validating the efficiency of our proposed approach. Secondly, we aim at analyzing the proposed approach over time analysis as opposed to state based (snapshot of bug repository) evaluation done here. Also, it is not known that whether accuracy of approach is more dependent on the terms extracted from bug reports or source code. Thus, we plan to conduct experimental evaluation whether such dependencies exists.

VII. REFERENCES

[1] N. Hoda, N. Narayan, B. Bernd, and H. Dina, "Bug Report Assignee Recommendation using Activity Profiles," in *Mining Software Repositories, 2013. MSR '13. 10th IEEE International Working Conference* On May 2013, pp. 22 –30.

[2] S. Ramin, A. John , K. Zarinah, and Z. Sima, "Automatic bug assignment using information extraction methods," in *International Conference on Advanced Computer Science applications and Technologies, 2012,* pp. 144-149.

[3] S. Ramin, A. John, K. Zarinah, and Z. Sima, "Why So Complicated? Simple term filtering and weighting for location based bug report assignment recommendation," in *Mining Software Repositories, 2013. MSR '13. 10th IEEE International Working Conference* on, May 2013, pp. 2 –11.

[4] D. Matter, A. Kuhn, and O. Nierstrasz, "Assigning bug reports using a vocabulary- based expertise model of developers," in *Mining Software Repositories, 2009. MSR '09. 6th IEEE International Working Conference* on, May 2009, pp. 131 –140.

[5] T. Zhang and B. Lee, "An automated bug triage approach: A concept profile and social network based developer recommendation," in Intelligent Computing Technology, ser. Lecture Notes in Computer Science, D.-S. Huang, C. Jiang, V. Bevilacqua, and J. Figueroa, Eds. Springer Berlin Heidelberg, 2012, vol. 7389, pp. 505–512.

[6] D. W. McDonald and M. S. Ackerman, "Expertise recommender: a flexible recommendation system and architecture," in *Proceedings of the 2000 ACM conference on Computer supported cooperative work, ser.* CSCW '00. New York, NY, USA: ACM, 2000, pp. 231–240.

[7] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, pp. 10:1–10:35, Aug. 2011

[8] P. Bhattacharya, and I. Neamtiu, "Fine grained Incremental Learning and Multi-feature tossing graphs to improve bug triaging," in 26th *IEEE International Conference on Software maintenance, 2010,* pp. 155-165.

[9] H. Hadi, N. Raymond , and G. Michael, " A market based bug allocation mechanism using predictive bug lifetimes" in *16th European Conference on Software maintenance and Reengineering, 2012,* pp. 149-157.

[10] L. Sangeeta, and S. Ashish, "Comparison of seven bug report types: A case study of google chrome browser project" in *19th Asia Pacific Software Engineering Conference, 2012,* pp. 517-526.

[11] G. Capobianco, A. D. Lucia, R. Oliveto, A. Panichella, and S. Panichella, "Improving ir-based traceability recovery via noun based indexing of software artifacts," *Journal of Software: Evolution and Process,* pp. n/a-n/a, 2012.

[12] W. Wu, W. Zhag, Y. Yang, and Q.Wang, "DREX: Developer recommendation with k- nearest neighbor search and expertise ranking," in *Software Engineering Conference(APSEC) 2011. 18th Asia Pacific,* dec. 2011, pp. 389-396.

[13] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *Proceedings of Seventh joint meeting of European Software Engineering Conference & ACM SIGSOFT symposium on Foundations of software engineering,* ser. ESEC/FSE '09. New York, NY, USA: ACM, 2009, pp. 111–120.

[14] http://9ol.es/porter_js_demo.html

