International Journal of System and Software Engineering
6 (1), June 2018, 01-09
http://www.publishingindia.com/ijsse

CLUBA: A Clustering-Based Approach for Bug Assignment

Mamdouh Alenezi[1*], Shadi Banitaan[2] and Mohammad Zarour[3]
[1]College of Computer & Information Sciences, Prince Sultan University, Saudi Arabia. Email: malenezi@psu.edu.sa
[2]Computer Science and Software Engineering, University of Detroit Mercy, USA. Email: banitash@udmercy.edu
[3]College of Computer & Information Sciences, Prince Sultan University, Saudi Arabia. Email: mzarour@psu.edu.sa
*Corresponding Author

Abstract: Nowadays, software systems are very complex which make software maintenance, especially bug fixing, very challenging. Identifying an appropriate developer to handle a new reported bug is very difficult and error-prone which results in a lengthy bug fixing process. In this paper, we propose a new developer recommendation approach, CLUBA, for assigning relevant developers to fix new bugs. The approach is based on clustering and it recommends a varying list of candidate developers based on their experience. The effect of choosing different percentages of terms from the corpus on clustering quality is carefully evaluated by applying one of the best feature selection methods. Then, similar bug reports are grouped together using the K-means clustering technique. After that, developers are ranked in each cluster based on their experience. We have validated CLUBA on four real open source projects and showed the feasibility of the approach by experimental evaluation.

Keywords: Bug assignment, Developer recommendation, Mining bug repositories, Software maintenance.

I. Introduction

Software repositories are available for most software projects. These repositories contain precious information about these software projects. The progress of software projects can be guided and managed by utilizing this information. Researchers have mined these software repositories to support software development and evolution (i.e., improve the design, refactoring, and maintenance). Bug Tracking System (BTS) is considered one of the important software repositories to guide maintenance activities. It allows both developers and users to submit bugs, propose enhancements, and comment on bug reports.

Bug fixing, one of the tasks in software maintenance, is an important laborious task. Bug triaging is an important software maintenance activity that involves assigning bug reports to experienced developers to fix them. However, it is usually performed manually. Manual bug triaging suffers from several problems. BTS usually receives a vast number of bug reports daily. For the Eclipse project, about 200 bugs are reported to its BTS daily near its release dates [5]. Each new reported bug must be assigned to an appropriate developer to fix it. In addition, a lot of bug reports were assigned incorrectly to developers. As a result, these reports need to be reassigned to other developers which lead to increasing the time of bug fixing.

In practice, due to the frequent changes of software development teams, it is difficult to identify an appropriate developer who has relevant experience in fixing similar bugs using manual triage process [3]. To solve these problems, some machine learning algorithms are employed to conduct automatic bug assignment [2, 3, 16, 26, 35]. Most of the bug assignment approaches are based on text categorization [3]. Most of these approaches predict only one developer. However, bug fixing is a collaborative process in which several developers contribute their knowledge and advice to solve the bug. Therefore, we propose CLUBA, an approach to automate the recommendation of developers who have the relevant experience to newly submitted bugs by utilizing the textural contents of bug reports and the comments made on them.

To summarize, we make the following key contributions in this work:

- We conduct a comprehensive study on using different term percentages to evaluate their effectiveness on improving the clustering performance.
- We formulate the bug assignment problem as a clustering task. The approach clusters similar bug reports and ranks experienced developers in each cluster based on two factors namely number of fixed bugs and number of contributed comments.
- We validate CLUBA on four real open source projects and show the feasibility of the approach by experimental evaluation.

The rest of the paper is organized as follows: Section 2 presents some background knowledge necessary to this work. Section 3 describes the CLUBA approach. The experimental evaluation and discussions are presented in Section 4. Section 5 discusses the threat to validity. Section 6 discusses related work. Section 7 concludes the paper.

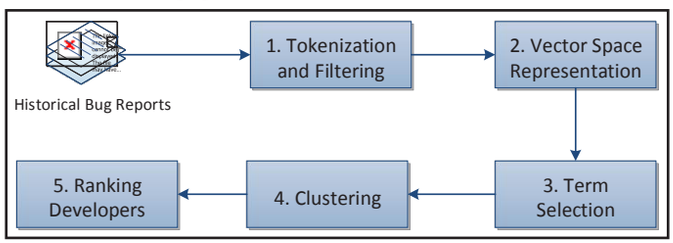


Fig. 1: The CLUBA Approach

II. Background

In this Section, we briefly introduce some background material related to CLUBA.

A. Clustering

Document clustering is one of the main text mining techniques that are widely used to summarize and organize text documents. Clustering organizes large documents into meaningful clusters. Document clustering is defined as follows: given a set of documents, we would like to partition them into a pre-specified or an automatically calculated number of clusters whereas documents in each cluster are more like each other than documents in other clusters. In other words, documents in one cluster share the same topic while documents in different clusters represent different topics [24]. The document feature space is usually of very high dimensionality. In addition, not all features are important for document clustering. Some of the features may be redundant or irrelevant and may even misguide the clustering result. Therefore, it is desirable to reduce the dimensionality of documents by applying feature selection techniques before clustering them. Feature selection not only reduces the high dimensionality of the feature space but also provides a better data understanding, which improves the clustering result [23].

B. Bug Reports

A bug report is made of several predefined fields, free-form text, attachments, comments, and dependencies. These predefined fields represent attributes of a bug. The text description of a bug report refers to the natural language contents, including the title of the bug report and a full description of the bug. To create a list of developers who helped in fixing a bug report, we extract from each report: the textual summary, the developer who fixed the bug and developers who added comments to the bug report.

C. Bug Report Comments

Bug tracking systems allow developers to give their feedback and comment on bug reports. Developers usually provide comments on bug reports they have experience in. On the same hand, it is common for developers to discuss the best ways to fixing a bug [19]. Almost all large software development teams work in a geographically distributed development environment in which these discussions are frequently reflected as comments in the bug report. Developers' comments are considered a vital major player in the life-cycle of a bug report, and most of them are related to how to resolve the bug. Those developers, who commented on the bug, often possess some expertise in fixing the bug [4].

III. The CLUBA Approach

CLUBA depends on the availability of an open bug repository for a software project for a period. The textual content of bug reports is unstructured and contains many irrelevant words provides an overview of CLUBA. CLUBA is divided into five main steps. The first step of CLUBA is to split the textual contents of bug reports into terms and filter out unnecessary terms. The second step in CLUBA is to create a corpus by representing bug reports as feature vectors where each feature represents a term. The third step is to choose distinct terms from the corpus by applying one of the best feature selection methods. The fourth step is to group similar bug reports together by using one popular partitioning algorithm. The fifth step is to rank developers in each cluster based on their experience. The following sections explain these steps in detail.

A. Tokenization and Filtering

In BTS, the textual contents of bug reports are represented by two separate fields namely summary and description. In this work, we use only the summary field as the textual representation of a bug report for twofold: 1) the description of bug reports has too many terms that are unrelated to the actual functionality of bug reports; 2) the summary of bug report consists of terms that are directly related to the functionality of bug report [20].

To reduce the size of the corpus, the set of the words can be reduced by tokenization and filtering. Tokenization is the isolation of word-like units from a text. It is a process in which text stream is breaking down into words called tokens. Filtering is the process of removing unnecessary tokens, tokens that do not add a new meaning to sentences they involve in. Filtering removes white-spaces, punctuation, numbers, and stop-words.

B. Vector Space Representation

After processing the textual content of bug reports, each bug report is represented by a vector of two dimensions: a term and its frequency. The terms are weighted by Term Frequency (TF) [25]. Despite its simple representation, the vector space model enables efficient analysis of documents and has been widely used in text mining approaches. Fig. 2 shows an example of a bug-term matrix (Corpus) representation.

	t_1	t_2	t_3	t_4
b_1	1	1	0	1
b_2	0	0	2	1
b_3	1	3	1	0
b_4	2	0	0	0
b_5	0	0	1	0

Fig. 2: An Example of a Bug-Term Matrix

C. Term Selection

Term selection methods are widely used to select the most representative terms from a corpus and to reduce

the dimensionality. It is also known to enhance document clustering [21, 22]. In this work, Chi-Square (X^2) is used to select distinctive terms from a corpus. X^2 is widely used to select features in document clustering [23, 22]. In statistics, the X^2 test is used to examine the independence of two events. The events, X and Y, are assumed to be independent if P(XY) = P(X)P(Y). In term selection, the two events are the occurrence of the term and the occurrence of the class. Terms are ranked with respect to the following equation [25]:

$$X^2(t,c) = \sum_{t\in\{0,1\}} \sum_{c} \frac{(N_{t,c} - E_{t,c})^2}{E_{t,c}} \quad (1)$$

Where, N is the observed frequency and E is the expected frequency for each state of term t and class c. X^2 is a measure of how much expected counts E and observed counts N deviate from each other.

D. Clustering

Bug assignment problem has been formulated previously as a classification task [3, 4]. In this work, we formulate the bug assignment problem as a clustering task. Clustering similar reports gives us more recommended developers than classification and allows for ranking these developers based on their collaborative effort in fixing these bugs. Although a bug report is fixed by only one developer, a group of developers has contributed their ideas and advice to fix the bug. Thus, this bug is fixed with the collaborative contribution made by the group of developers (commenters). This phase compromises the following steps:

1. Group (cluster) similar bug reports together.
2. Determine the developers who fixed these bugs.
3. Assign a newly coming bug reports to one of the groups.
4. Recommend potential developers from that group to fix it.

The first step in this phase is to cluster bug reports based on their textual similarity. Bug reports in each cluster are more similar to each other than bug reports in different clusters. In other words, each cluster represents a topic or a bug category. The second step is to determine the active developers in that group by examining the assigned to field in these bug reports. Since these developers have fixed similar bug reports, they are considered qualified to fix new bug reports similar to the bugs in this group. The third step is to assign a newly coming bug report to the closest cluster by measuring the similarity between the bug report and clusters. The last step is to rank developers in each group based on their experience, then assign a newly coming bug report to the top-ranked candidate developers. For clustering, we use the K-means clustering technique. The following section discusses the clustering approach used in this work.

i. K-Means Clustering

There are two main clustering categories namely hierarchical and partitioning methods. We choose one of the partitioning methods (K-means) for the following reasons: 1) the partitioning clustering methods are known to be very suitable for clustering large documents because of their low computational requirements [11, 41]; 2) hierarchical methods do not provide the reallocation of documents which may have been wrongly classified in the early phases of the text analysis [15]; and 3) K-means performs better than hierarchical clustering in document clustering [29].

K-means clustering is a technique commonly used to automatically partition a dataset into K groups in which each instance belongs to the cluster with the nearest mean. The first step in K-means is to select K bug reports as centers. Then, each instance (bug report) is assigned to its closest cluster center. After that, each cluster center is updated to be the mean of its constituent instances. The algorithm converges when there is no further change in assignment of instances to clusters. In K-means, the number of clusters (K) needs to be specified in advance. K-means method is known to be very sensitive to K, therefore K should be selected carefully. There is a number of methods of how to specify K. To specify the number of clusters (K), we use the following commonly used equation [1]:

$$K = \frac{m \times n}{t} \quad (2)$$

Where, m is the number of bug reports, n is the number of terms, and t is the number of non-zero entries in the bug-term matrix. The idea here is that if the number of nonzero entries is increased, the similarity among bug reports is increased which leads to a smaller number of clusters.

In K-means, a distance measure should be defined in order to measure the similarity between the bug report and the cluster centers in an iterative fashion. The Cosine similarity is widely used for that purpose [13, 14] and it is defined as follows:

$$Cosine(A,B) = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}} \quad (3)$$

Where, A and B represent the vector space representation of two bug reports. The resulting similarity ranges from 0 to 1, since the term frequencies are only positive.

ii. Ranking Developers

After clustering similar bug reports into K clusters, we need to rank developers in each cluster based on their collaborative effort on fixing these bug reports. Two factors are considered to rank developers namely number of fixed bugs (i.e., number of bug reports in the cluster that assigned to that developer) and number of contributed comments (i.e., number of comments made by that developer on bug reports in that cluster). The following equation represents the ranking criteria:

$$\alpha \times \text{fixed bugs} + (1 - \alpha) \times \text{contributed comments} \quad (4)$$

The parameter α is introduced in order to investigate which factor should be given more weight.

IV. Experiments and Discussion

In this Section, we report the results of applying the CLUBA approach on real datasets. A description of the datasets used in this work is presented in Section A Results and discussions are shown in Section B.

A. Datasets and Prepossessing

We evaluate CLUBA based on bug repositories of Mandriva[1], Freedesktop[2], NetBeans[3], and Samba[4]. We collect the bug reports that have the status of [Closed, Verified, and Resolved] and the resolution of [Fixed]. For each bug report, we extract the bug ID, the assignee, the summary, and the commenters. For Mandriva, we choose 9199 bug reports from April 1^{st}, 2008 until December 25^{th}, 2012. For Freedesktop, we choose 9981 bug reports from January 1^{st}, 2011 until December 25^{th}, 2012. For NetBeans, we choose 7444 bug reports from January 1^{st}, 2012 until December 25^{th}, 2012. For Samba, we choose 2114 bug reports from January 1^{st}, 2008 until December 25^{th}, 2012. Table I shows statistics about the selected datasets.

Table I: Statistics About the Datasets

Name	# Bugs	From	To
Mandriva	9199	Apr, 01, 2008	Dec 25, 2012
Freedesktop	9981	Jan, 01, 2011	Dec 25, 2012
NetBeans	7444	Jan, 01, 2012	Dec 25, 2012
Samba	2114	Jan, 01, 2008	Dec 25, 2012

We want to refine the training set further to remove reports that are assigned to inactive developers (i.e., developers who no longer work on the project or developers who have only fixed a small number of bugs) or reports that do not have sufficient words to describe a meaningful description. We eliminate bug reports that have less than four words since they give no meaningful description and too short to hold relevant information. To determine active developers, we consider developers who have fixed at least 15 bug reports in the dataset. Regarding comments, we exclude developers who only commented once on a bug report. We consider this as a noisy source of data. We divide the data into training and testing in which bug reports in the last four months are considered as testing instances. Table II shows a summary of the refined datasets.

Table II: Summary of the Datasets

Name	Training	Testing	# Developers
Mandriva	7862	8	99
Freedesktop	7074	1016	113
NetBeans	3070	2661	50
Samba	1703	172	16

[1]https://qa.mandriva.com/
[2]https://bugs.freedesktop.org/
[3]http://netbeans.org/bugzilla/ [4]https://bugzilla.samba.org/

B. Results

This Section presents the results of CLUBA and discusses them. The clustering results are presented in Section i and the bug assignment results are shown in Section ii.

i. Clustering Results

We first investigate the effect of using different number of terms obtained by X^2 on the performance of clustering. Different percentages of the terms (10% to 50%) are considered. Tables III, IV, V and VI show the clustering results of Mandriva, Freedesktop, NetBeans, and Samba respectively where ADC denotes the average number of developers per cluster, ACS denotes the average cluster size, TW represents the total within-cluster sum of squares, BE represents the between-cluster sum of squares, and K represents the number of clusters. The results indicate that the best results are achieved when the percentage of terms is 10% for all datasets. TW is insignificant compared to BE which indicates that bug reports in each cluster are similar while bug reports in different clusters are not.

Table III: Mandriva Clustering Results

# Terms	ADC	ACS	TW	BE	K
All Terms	3.43	4.59	18225.83	21957.15	1713
10%	3.08	10.16	1105.67	7258.06	774
20%	2.77	6.64	1957.35	9636.96	1184
30%	2.34	5.53	2860.26	11206.18	1421
40%	2.65	5.37	5044.99	13056.55	1465
50%	3.33	5.93	8785.86	15006.91	1326

Table IV: Freedesktop Clustering Results

# Terms	ADC	ACS	TW	BE	K
All Terms	2.49	3.84	17393.28	21640.25	1841
10%	2.60	8.27	1024.76	6672.84	856
20%	2.39	5.67	2130.14	9092.10	1247
30%	2.55	4.99	3666.36	11198.91	1416
40%	2.43	4.85	5587.35	12765.93	1459
50%	3.05	5.12	8935.10	14549.49	1381

ii. Bug Assignment Results

Each dataset is divided into training and testing. The training set is clustered into several groups. Then each bug report in the testing set is assigned to potentially experienced developers after it is assigned to one of the clusters. We investigate the effect of predicting different number of developers ranging from one to five on the bug assignment accuracy.

TABLE V: NETBEANS CLUSTERING RESULTS

# Terms	ADC	ACS	TW	BE	K
All Terms	2.20	2.89	5531.55	11709.84	1064
10%	2.52	6.87	756.85	3359.07	447
20%	2.44	5.08	1342.84	4638.76	604
30%	2.30	4.18	1722.58	5548.06	734
40%	2.41	3.86	2304.30	6487.08	796
50%	2.42	3.68	2940.72	7498.37	835

TABLE VI: SAMBA CLUSTERING RESULTS

# Terms	ADC	ACS	TW	BE	K
All Terms	1.90	2.88	3515.45	5674.97	592
10%	1.56	5.05	117.32	1471.20	337
20%	1.49	3.69	278.29	2033.84	461
30%	1.60	3.45	560.99	2669.78	493
40%	1.95	3.80	1300.01	3276.38	448
50%	2.13	3.95	1991.54	3646.46	431

$$\text{Precision} = \frac{\text{Recomm. Developers} \cap \text{Actual Developers}}{\text{Recomm. Developers}} \tag{5}$$

$$\text{Recall} = \frac{\text{Recomm. Developers} \cap \text{Actual Developers}}{\text{Actual Developers}} \tag{6}$$

$$\text{F-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{7}$$

For measuring the accuracy of bug assignment using CLUBA we use Precision, Recall, and F-score. Precision is the percentage of recommended developers who truly helped in fixing the bug report. Recall is the percentage of developers who helped on solving the bug who were truly suggested. F-score represents the harmonic mean of Precision and Recall.

Fig. 3 shows the F-score of bug assignment where the x-axis represents varying α from 1 to 0.4 and the y-axis represents the F-score with different number of recommended developers. For Mandriva, the best F-score (0.571) is obtained when α equals to 1 and the number of predicted developers equals to 1. When α is 1, the F-score gives the best results. For Freedesktop, the best F-score (0.211) is obtained when α equals to 0.9 and the number of predicted developers equals to 3. The best F-score is obtained when α is 0.9 for all number of predicted developers except when predicting one developer. For NetBeans, the best F-score (0.179) is obtained when α equals to 0.9 and the number of predicted developers equals to 2. The best F-score is obtained when α equals to 0.9. For Samba, the best F-score (0.259) is obtained when α equals to 0.9 and the number of predicted developers equals to 5. The best F-score is obtained when α equals to 0.8 for all except when predicting 5 developers.

Fig. 3 indicates that the best results are obtained when α ranges from 0.8 to 1 which give an evidence that both factors contribute in ranking developers. In addition, the number of fixed bugs factor should be given more weight (i.e., when α = 0.9, the weight for the number of fixed bugs factor is 0.9 while the weight for the number of contributed comments factor is 0.1). Furthermore, predicting 2 or 3 candidate developers shows a promising results.

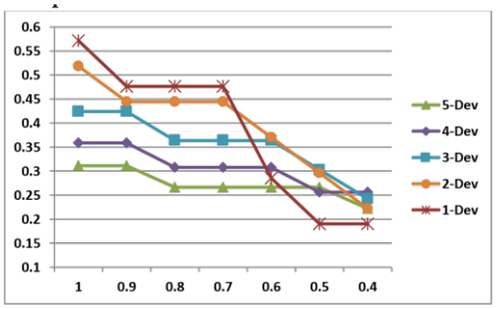
V. THREATS TO VALIDITY

In this work, we only use the textual representation to represent bug reports; other meta-data are available and can be used to represent bugs such as priority and severity. As we mentioned before, K-means is very sensitive to the number of clusters. In our work, we fixate *K* but in future work, we will investigate the effect of using different *K*s on the results. Bug report's comments are considered a noisy source of data. In order to remove casual developers, we removed commenters with only one comment. We use this as indication that they do not have prominent levels of activity and are not key members of the collaborative group. While this means that we remove some members from our analysis, their low collaborative participation implies that these participants have little impact. We have applied CLUBA on open source bug repositories only; commercial projects may have different characteristics that may require some changes. In addition, we only use Bugzilla bug reports. Other bug tracking systems are available such as Gnats, Trac and Jira that model bug reports differently. Therefore, CLUBA should be applied on other projects in order to generalize the results.

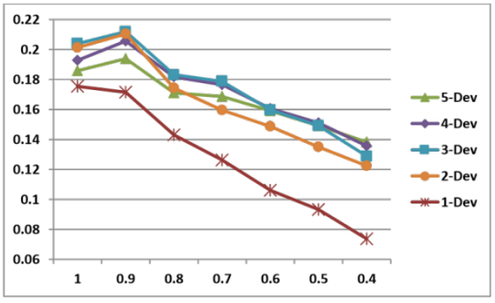
VI. RELATED WORK

Bug assignment and triaging is a time-consuming and error-prone activity, and can significantly affect project duration. Projects' managers need to assess developers competency and productivity level after assessing the bug validity before deciding to assign the bug to a certain developer. As the project size increases bug triaging becomes a more complex process with critical effects on the project's duration. Bug triage has been for a long time been done using traditional manual approach. Nowadays, machine learning algorithms and social network analysis can be used to automate or semi-automate the bug triage process.

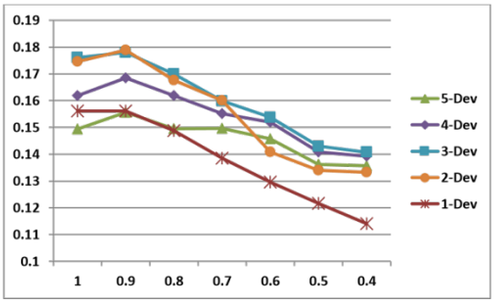
Automatic bug triage can reduce the probability of re-assignment and reduce triagers time by recommending the most appropriate assignees [38]. Numerous studies have been carried out to find candidate developers for resolving new bug reports. According to the different techniques, these studies have been classified in [38] into five category, namely, machine learning-based recommender, expertise model-based recommender, tossing graph-based recommender, social network-based recommender and topic model-based recommender.



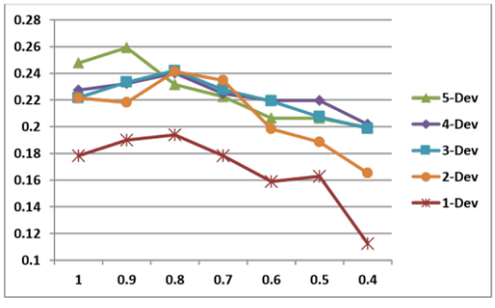
(a) Mandriva



(b) Freedesktop



(c) NetBeans



(d) Samba

Fig. 3: F-Score by Varying α with Different Number of Recommended Developers

A. Machine-Learning-Based Recommender

Machine learning techniques are proven to be useful is various domains for better decision making, this includes bug assignment decisions. Cubranic and Murphy [10] proposed to use machine learning techniques to assist in bug triage. They used supervised Bayesian learning to train a classifier with the textual content of resolved bug reports. Then this classifier is used to classify newly incoming bug reports with accuracy rate of 30%, considering Eclipse as a case study.

Anvik *et al.* [3, 6] have employed several different machine-learning algorithms, e.g., NaiveBayes, C4.5 and SVM, to recommend a list of appropriate developers for fixing a new bug and improve the performance of automatic bug triage. They were able to improve the precision by up to 97% for Eclipse and 70% for Firefox by utilizing component-based developer recommender.

Xuan *et al.* [35] proposed a semi-supervised learning approach with a weighted recommendation list for bug triage to solve the problem of low quality of bug reports. Their approach improved the classification accuracy of bug triage by up to 6% on the Eclipse project.

Xia *et al.* [33] proposed DevRec approach for recommending the bug fixers. DevRec combined two kinds of analysis, including bug reports-based analysis and developer based analysis. Based on Multi-Label KNN, bug report based analysis can find the k-nearest bug reports to a new given bug by using the features (i.e., terms, product, component and topics) of bug reports. For developer based analysis, the distance between a developer and a selected feature is measured.

Anvik and Murphy [4] have presented two approaches for determining who has the implementation expertise for a bug report by utilizing data from two types of repositories: The source repository check-in logs and The bug repository. They have found that different repositories are useful in different situations, based on what is required.

Matter *et al.* [26] represented a developer's expertise using the vocabulary found in the developer's source code. They recommend experienced developers by extracting information from new bug reports and looking it up in the vocabulary. Their approach does not require a history of bug reports and was tested on 130,769 Eclipse bug reports. They achieved 33.6% top-1 precision and 71.0% top-10 recall using eight years of Eclipse project. On the other hand, Banitaan and Alenezi [7] proposed TRAM, an approach with the aim of increasing the prediction accuracy of bug triage by using the most discriminated terms of bug reports, the components in which the bugs belong to, and the reporter who filed the bug. Their experimental evaluation showed that TRAM outperforms many existing machine learning-based approaches in terms of classification accuracy.

B. Expertise Model-Based Recommender

A model that captures the developer's expertise based on his historical bug-fixing data is developed. For instance, [26] has built the expertise model based on the vocabularies used by developers and documented in their source code files. The terms appearing in the bug report are then compared to the developed expertise model and accordingly discover or recommend the suitable developer. Similarly, fuzzy set and cache-based modelling of the bug-fixing expertise of developers has been

developed in [30]. A ranked list of developers, in terms of expertise in specified map locations, is developed in [28] where the code changes history and the location of bugs are used to build the expertise model.

C. Tossing Graph-Based Recommender

Developer networks can be used to discover team structure where it can find suitable assignees for a bug-fixing task. Such network is called a tossing graph that captures bug tossing history and introduced in [17]. Several studies, later on, tried to improve the accuracy of bug triage and reduced tossing path lengths and tested the resulted improvements on Mozilla and Eclipse where results showed great improvement, for more information refer to [12, 8, 9].

D. Social Network-Based Recommender

Social network technique has been used to find the potentially experienced assignees for fixing each new reported bug in [18]. Using social networks is a convenient way to analyze the relationship between developers in the bug-fixing process. The developers experience on fixing bugs can be understood based on the analysis of the commenting activities among developers, hence the candidate assignees for a new bug can be found. Several studies have used the same technique to assign bugs. For instance, Zhang and Lee [37] proposed a developer recommendation method for assigning appropriate developers to fix bugs. Their approach was based on social network and experts' feedback to recommend some candidate developers. They considered the fixing efficiency and the experience of these candidate developers to rank them. They achieved F-score of 40% when the candidate list is 3. For more examples, refer to [36, 31].

E. Topic Model-Based Recommender

Topic-model recommender works to find historical bug reports similar to a new bug report, which share the same topic(s). The scholars expect to introduce a topic model for improving the accuracy of automatic bug triage. Studies [34, 27, 39] adopted topic model to recommend the best developers for fixing the given bugs. Zhang *et al.* [40] modeled developer's interest in a topic based on their comments. They proposed an approach with the name of En-LDA that recommends a ranked list of developers who are potentially appropriate to handle the bug. Xia *et al.* [32] proposed a specialized topic modeling algorithm for bug triaging. In their work, they proposed an incremental learning method to assign an appropriate fixer to the newly reported bug.

VII. Conclusions

This paper proposed, CLUBA, an approach to recommend highly experienced developers to contribute to resolving newly submitted bug reports. After clustering bug reports based on their distinctive terms, CLUBA ranks candidate developers based on their collaborative effort on fixing bug reports. For newly coming bug report, CLUBA assigns it to a relevant cluster and recommends potential developers from that cluster to solve it. Experimental evaluations were conducted on four datasets. The clustering results indicated that the best fitness criteria are achieved when the percentage of terms is 10% for all datasets. We investigated the use of two different factors in ranking developers. Both factors contributed unequally in ranking developers where the contribution of the number of fixed bugs factor is significantly larger than the contribution of the number of contributed comments factor.

In future, we will examine the performance of CLUBA on other bug repositories in order to generalize the results. In addition, we would like to investigate the effect of using different similarity measures on clustering results. Moreover, we would like to examine the effect of utilizing other ranking criteria such as the average fixing time.

References

[1] A. A. Al-Subaihin, F. Sarro, S. Black, L. Capra, M. Harman, Y. Jia, and Y. Zhang, "Clustering mobile apps based on mined textual features," In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, p. 38, ACM, 2016.

[2] J. Anvik, "Automating bug report assignment," In *Proceedings of the 28th International Conference on Software Engineering*, pp. 937-940, ACM, 2006.

[3] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?," In *Proceedings of the 28th International Conference on Software Engineering*, pp. 361-370, ACM, 2006.

[4] J. Anvik, and G. C. Murphy, "Determining implementation expertise from bug reports," In *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops, 2007)*, pp. 2-2, IEEE, 2007.

[5] J. Anvik, L. Hiew, and G. C. Murphy, "Coping with an open bug repository," In *Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange*, pp. 35-39, ACM, 2005.

[6] J. Anvik, and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 20, no. 3, 2011.

[7] S. Banitaan, and M. Alenezi, "Tram: An approach for assigning bug reports using their metadata," In *2013 Third International Conference on Communications and Information Technology (ICCIT)*, pp. 215-219, IEEE, 2013.

[8] P. Bhattacharya, and I. Neamtiu, "Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging," In *2010 IEEE International Conference on Software Maintenance (ICSM),* pp. 1-10, IEEE, 2010.

[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] D. Cubrani´c, and G. C. Murphy, "Automatic bug triage using text categorization," In *SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering*, pp. 92-97, Citeseer, 2004.

[11] I. S. Dhillon, "Co-clustering documents and words using bipartite spectral graph partitioning," In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 269-274, ACM, 2001.

[12] N. Friedman, I. Nachman, and D. Pe´er, "Learning bayesian network structure from massive datasets: The sparse candidate algorithm," In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pp. 206-215, Morgan Kaufmann Publishers Inc., 1999.

[13] J. Ghosh, and A. Strehl, "Similarity-based text clustering: A comparative study," *Grouping Multidimensional Data,* pp. 73-97, Springer, Berlin, Heidelberg, 2006.

[14] A. Huang, "Similarity measures for text document clustering," In *Proceedings of the Sixth New Zealand Computer Science Research Student Conference (NZCSRSC2008),* pp. 49-56, Christchurch, New Zealand,

[15] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM Computing Surveys (CSUR)*, vol. 31, no. 3, pp. 264-323, 1999.

[16] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pp. 111-120, ACM, 2009.

[17] 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.

[18] D. Kempe, J. M. Kleinberg, and E. Tardos, "Maximizing the spread of influence through a social network," *Theory of Computing*, vol. 11, no. 4, pp. 105-147, 2015.

[19] A. J. Ko, R. DeLine, and G. Venolia, "Information needs in collocated software development teams," In *Proceedings of the 29th International Conference on Software Engineering*, pp. 344-353, IEEE Computer Society, 2007.

[20] A. J. Ko, B. A. Myers, and D. H. Chau, "A linguistic analysis of how people describe software problems," In *Proceedings of the Visual Languages and Human-Centric Computing (VLHCC'06)*, pp. 127-134, IEEE Computer Society, Washington, DC, USA, 2006.

[21] Y. Li, C. Luo, and S. M. Chung, "Text clustering with feature selection by using statistical data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 5, pp. 641-652, 2008.

[22] L. Liu, J. Kang, J. Yu, and Z. Wang, "A comparative study on unsupervised feature selection methods for text clustering," In *2005 IEEE International Conference on Natural Language Processing and Knowledge Engineering (NLPKE'05)*, pp. 597-601, IEEE, 2005.

[23] T. Liu, S. Liu, Z. Chen, and W. Y. Ma, "An evaluation on feature selection for text clustering," In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pp. 488-495, 2003.

[24] C. Luo, Y. Li, and S. M. Chung, "Text document clustering based on neighbors," *Data and Knowledge Engineering*, vol. 68, no. 11, pp. 1271-1288, 2009.

[25] C. D. Manning, P. Raghavan, and H. Schu¨tze, *Introduction to Information Retrieval*, vol. 1, Cambridge University Press, Cambridge, 2008.

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

[27] H. Naguib, N. Narayan, B. Bru¨gge, and D. Helal, "Bug report assignee recommendation using activity profiles," In *10th IEEE Working Conference on Mining Software Repositories (MSR'13),* pp. 22-30, IEEE, 2013.

[28] F. Servant, and J. A. Jones, "Whosefault: Automatic developer-to-fault assignment through fault localization," In *Proceedings of the 34th International Conference on Software Engineering*, pp. 36-46, IEEE Press, 2012.

[29] M. Steinbach, G. Karypis, and V. Kumar, "A comparison of document clustering techniques," In *KDD Workshop on Text Mining*, vol. 400, pp. 525-526, Boston, 2000.

[30] A. Tamrawi, T. T. Nguyen, J. M. Al-Kofahi, and T. N. Nguyen, "Fuzzy set and cache-based approach for bug triaging," In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, pp. 365-375, ACM, 2011.

[31] W. Wu, W. Zhang, Y. Yang, and Q. Wang, "Drex: Developer recommendation with k-nearest-neighbor search and expertise ranking," In *2011 18th Asia-Pacific Software Engineering Conference (APSEC)*, pp. 389-396, IEEE, 2011.

[32] X. Xia, D. Lo, Y. Ding, J. M. Al-Kofahi, T. N. Nguyen, and X. Wang, "Improving automated bug triaging with specialized topic model," *IEEE Transactions on Software Engineering*, vol. 43, no. 3, pp. 272-297, 2017.

[33] X. Xia, D. Lo, X. Wang, and B. Zhou, "Accurate developer recommendation for bug resolution," In *2013 20th Working Conference on Reverse Engineering (WCRE)*, pp. 72-81, IEEE, 2013.

[34] X. Xie, W. Zhang, Y. Yang, and Q. Wang, "Dretom: Developer recommendation based on topic models for bug resolution," In *Proceedings of the 8th International Conference on Predictive Models in Software Engineering*, pp. 19-28, ACM, 2012.

[35] J. Xuan, H. Jiang, Z. Ren, J. Yan, and Z. Luo, "Automatic bug triage using semi-supervised text classification," In *Proceedings of International Conference on Software Engineering and Knowledge Engineering (SEKE'10)*, pp. 209-214, 2010.

[36] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," In *2012 34th International Conference on Software Engineering (ICSE)*, pp. 25-35, IEEE, 2012.

[37] T. Zhang, and B. Lee, "How to recommend appropriate developers for bug fixing?," In *2012 IEEE 36th Annual Computer Software and Applications Conference (COMPSAC)*, pp. 170-175, IEEE, 2012.

[38] T. Zhang, H. Jiang, X. Luo, and A. T. S. Chan, "A literature review of research in bug resolution: Tasks, challenges and future directions," *The Computer Journal*, vol. 59, no. 5, pp. 741-773, 2016.

[39] T. Zhang, G. Yang, B. Lee, and E. K. Lua, "A novel developer ranking algorithm for automatic bug triage using topic model and developer relations," In *2014 21st Asia-Pacific Software Engineering Conference (APSEC)*, vol. 1, pp. 223-230, IEEE, 2014.

[40] W. Zhang, Y. Cui, and T. Yoshida, "En-lda: An novel approach to automatic bug report assignment with entropy optimized latent dirichlet allocation," *Entropy*, vol. 19, no. 5, p. 173, 2017.

[41] Y. Zhao, and G. Karypis, "Empirical and theoretical comparisons of selected criterion functions for document clustering," *Machine Learning*, vol. 55, no. 3, pp. 311-331, 2004.