2011 18th Asia-Pacific Software Engineering Conference

DREX: Developer Recommendation with K-Nearest-Neighbor Search and Expertise Ranking

Wenjin Wu, Wen Zhang, Ye Yang, Qing Wang
Laboratory for Internet Software Technologies
Institute of Software, Chinese Academy of Sciences
Beijing 100190, P.R.China
{wuwenjin,zhangwen,ye,wq}@itechs.iscas.ac.cn

***Abstract*—This paper proposes a new approach called DREX (Developer Recommendation with k-nearest-neighbor search and EXpertise ranking) to developer recommendation for bug resolution based on K-Nearest-Neighbor search with bug similarity and expertise ranking with various metrics, including simple frequency and social network metrics.**

We collect Mozilla Firefox open bug repository as the experimental data set and compare different ranking metrics on the performance of recommending capable developers for bugs. Our experimental results demonstrate that, when recommending 10 developers for each one of the 250 testing bugs, DREX has produced better performance than traditional methods with multi-labeled text categorization. The best performance obtained by two metrics as Out-Degree and Frequency, is with recall as 0.6 on average. Moreover, other social network metrics such as Degree and PageRank have produced comparable performance on developer recommendation as Frequency when used for developer expertise ranking.

***Index Terms*—Open Bug Repository, Developer Recommendation, KNN Search, Expertise Ranking.**

I. Introduction

Open source software projects or large commercial software projects usually adopt a bug tracking system such as Bugzilla, GNATS, JIRA and Mantis, etc., to deal with bugs occurring in the projects. With this kind of system, users and developers can report bugs encountered in using or developing the software and consequently, large amount of bug reports are submitted to the open bug repository of bug tracking system by software users and developers. To make the software with high quality and adequate for real use, those (at least some of) bugs should be resolved before the release of software.

However, a difficult problem confronted with open bug repository is that large number of bug reports are submitted to the bug tracking system every day resulting in an overloaded burden for developers to find bug reports of their interest or within their expertise. For instance, about 200 bugs are reported to Eclipse bug tracking system daily near its release dates [1], and Debian has about 150 bugs daily [2]. With the goal of resolving those incoming deluge of new bug reports in OSS projects such as Eclipse and Mozilla, there are certain developer roles, namely bug report triagers, whose task is to diagnose submitted bug reports and assign each of them to the developers of the software project who have expertise in or responsibility for these bugs.

Usually, human triagers decide to whom a new bug report should be assigned by looking at the information of historical bugs resolved by the developers. Nevertheless, due to limited knowledge of the cause of those bugs and expertise of every developer in the whole software project, it is impossible for triagers to precisely match developers with bugs of their interest or potential expertise manually. Even worse, in some OSS projects without bug report triagers, developers need to skim or read through all the bug reports to decide whether or not some bug reports exist within their interest, which is an overwhelming burden for them. Thus, manual handling of bug reports often bring about inaccurate bug assignment and time delay in resolving bugs.

The goal of our study is to recommend bugs to developers who have the potential to contribute to bug resolution. Bug tracking system is a platform of bug resolution through developers' coordination [3]. Dan Bertram et al. [4] maintain that bug tracking is fundamentally a social process. Although a bug report was "fixed" (or the fix was committed to the source code version system) by only one developer finally, a group of developers have contributed their ideas and advices in process of fixing the bug. Thus, we may regard that this bug was "fixed" with the collaborative contribution made by the group of developers. For instance, on "*bug_id=333160 Document the Recently Closed Tabs menu and the keyboard shortcut*"(https://bugzilla.mozilla.org/show_bug.cgi?id=333160)of Mozilla Firefox, 5 developers discussed on fixing the bug and they posted 19 comments in the discussion. In this case, "Ronny Perinke" reported this bug. Then, "Adam Guthrie", "Hasse", "Reed Loden" and "Steffen Willberg" joined the discussion on how to fix this bug. Finally, "Steffen Willberg" fixed this bug.

This paper studies the problem of automatic recommendation of developers to bug reports to which the developers may contribute potential knowledge. By utilizing the information of historical bug reports and its following comments in open bug repository, we proposes a new approach called DREX to developer recommendation and DREX comprises two components. The first component is to search similar historical bug reports within its K-nearest neighbors and the second component is to rank developers' expertise using developers' participation records in discussing the similar bug reports. Here, Frequency and social network metrics are adopted to rank the developers'

1530-1362/11 $26.00 © 2011 IEEE
DOI 10.1109/APSEC.2011.15

expertise on resolving the new bug reports. Note that we are studying a different problem from bug assignment [5] [6] [7] [8]. While bug assignment attempts to find one or some developers who can "fix" a given reported bug, our work is to recommend a group of developers who may have interest or potential knowledge on resolving the reported bug.

The contribution of this paper includes two aspects. First, we propose a new approach called DREX to developer recommendation using K-Nearest- Neighbor search for similar bug reports and various metrics to rank developers' expertise. Second, we compare DREX and developer recommendation based on text categorization. Also, we adopt various metrics for developers' expertise ranking, including simple Frequency and other six social network metrics as Indegree, Outdegree, Degree, PageRank, Betweenness and Closeness.

The remainder of this paper is organized as follows. Section II presents the background knowledge of developer recommendation. Section III proposes DREX to developer recommendation. Section IV describes the experiments on Mozilla Firefox bug data and explains the experimental results. Section V discusses the threads to validity of DREX. Section VI describes related work and Section VII concludes this paper.

II. Background and Definitions

A. Bug Reports Related

Generally, a bug report comprises many predefined meta-fields and free-form textual contents. Predefined meta-fields describe the basic attributes of the bug such as "report_id", "reporter", "product_id", "component_id", etc. For instances, "product_id" stands for the product in which the bug occurred, and "component_id" stands for the specific component of a product in which the bug was found. Freeform textual contents of a bug report refer to the natural language description of the bug that actually includes two parts. The first part is the content of the bug report (for example, the "short_desc" in the table "bugs" in bugzilla database) and the second part contains the comments of the bug report posted by the developers who are interested in this bug. Figure 1 shows the details extracted from the Mozilla bug report with "bug_id" number as 342,133 (https://bugzilla.mozilla.org/show_bug.cgi?id=342133).

In bug tracking system, like Bugzilla, the content of "Description", which was recorded in the "thetext" field of "logdescs" table in Bugzilla database, is the main body of the bug report that usually includes some parts of relevant information describing the bug such as steps to reproduce the actual results, expected results, building data, platform, additional builds and platforms, crushing information, etc. Comments are generated by either developers' manual appending or automatic appending by bug the tracking system induced by developers' manipulation on the bug.

Besides "Description" submitted by the reporter, developers' comments also play an important role in the whole life cycle of a bug report, and most of them are relevant with how to fix the bug or the consequences of a particular fix happened [9]. These comments are presented by developers in a distributed development environment such as open source

Fig. 1. The detailed information of the Mozilla bug report with "bug_id" number as 342,133. The texts above "Description" are the predefined meta-fields and the texts below the "Description" are free-form textual contents of the bug report.

software community to discuss the solution of the reported bug. Those developers, who present comments on the bug, often have some interest in this bug and possess some expertise those are useful to resolving the bug [10]. Automatic generated comments happen when developers submit a patch, upload an attachment or mark some other bugs as a duplicate of this bug report and these automatic comments are usually followed by manual appending.

B. Document Representation for Bug Reports

Natural language processing (NLP) is employed in this study to transfer the bug reports and developers' comments into numeric vectors which can be processed by K-Nearest-Neighbor search. To proceed, we pose two definitions: document for historical bug report and document for new bug report.

Definition 2.1: Document for historical bug report. For a historical bug report, we combine its description br_i and the following comments $c_{br_i} = (c_{i,1}, ..., c_{i,|c_{br_i}|})$ to compose a document d_i. Thus, we call d_i is the document for historical bug report br_i.

Definition 2.2: Document for new bug report. For a new bug report, we use its description br_{new} to compose a document d_{new}. Thus, we call d_{new} is the document for the new bug report br_{new}.

Actually, document representation includes two procedures: document indexing and term weighting. Tokenization, stop word elimination and stemming are often employed to preprocess the documents of both historical and new bug reports and, vector space model is employed to index the document contents using the terms occurring in the documents [11] [12]. For instance, the 100 stop words from USPTO (United States Patent and Trademark Office) patent full-text and image database (online: http://ftp.uspto.gov/patft/help/stopword.htm)

can be used for stop word elimination. Porter stemming algorithm is often used for English word stemming processing and it can be downloaded freely from the Internet (online: http://tartarus.org/~martin/PorterStemmer/).

In term weighting, TF*IDF [13] described in Equation 1 is often used for this task. Here, $w_{i,j}$ is the weight for term i in document j, the number of documents is the same as the number of historical bug reports derived from the open bug repository, $tf_{i,j}$ is the term frequency of term i in document j and df_i is the document frequency of term i in the whole document collection. After the document representation of bug reports and their comments, each document d_j is represented as $d_j = (w_{1,j}, ..., w_{i,j}, ..., w_{n,j})$ where n is the total number of terms in the document collection.

$$w_{i,j} = tf_{i,j} \times log(\frac{\#\ of\ documents}{df_i}) \quad (1)$$

C. K-Nearest-Neighbor Search

K-Nearest-Neighbor (KNN) classification is a type of instance-based lazy learning that classifies objects based on the top K similar training objects in the feature space [14]. In this study, we adopt the key idea of KNN classification to search the top K similar historical bug reports of a new bug report (they are represented as document vectors in Section II.B) to establish the similar bug set of the new bug report. The cosine similarity shown in Equation 2 is used to measure the similarity of two document vectors where d_{new} denotes the document vector of a new bug report and d_{his} denotes the document vector a historical bug report. In this study, KNN search is used to search the existing documents of historical bug reports based on their similarities with the new bug br_{new}.

$$sim(d_{new}, d_{his}) = \frac{d_{new} \cdot d_{his}}{|d_{new}||d_{his}|} \quad (2)$$

D. Social Network Analysis

We noticed that in OSS project (see Section VI for related work), bug resolution is a collaborative work within a group of developers and social communities of developers with similar interest are arising from their frequent corporation. Motivated by this observation, we introduce social network analysis here to discover those developers of great influence in developers' communities for bug resolution.

Social network analysis [15] is originated in sociology and usually used to analyze the complex sets of relationships between members of social systems, with the goal of explaining social phenomena. The nodes in the social network being studied are a set of developers and the links of a social network are a set of relationships of these developers in contributing bug resolution.

An essential function of social network analysis is to rank the importance of the developers according to their positions in the network [16]. To accomplish this goal, centrality indices are defined on the nodes of the graph, such as closeness centrality, graph centrality, betweenness centrality, etc. High centrality of a node means its relative high importance to other nodes. In this study, four metrics as degree centrality, in-degree centrality and out-degree centrality, PageRank, betweenness centrality and closenesss centrality are used to rank the developers' expertise. Degree centrality is defined as the number of links incident upon a node (i.e., the number of ties that a node has). Betweenness centrality is the number of shortest paths from all vertices to all others that pass through that node. Closenesss centrality is defined as the mean geodesic distance (i.e., the shortest path) between a vertex v and all other vertices reachable from it. PageRank [17] is a variant of eigenvector centrality measure proposed by Sergey Brin and Larry Page, assigns a numerical weighting to each element of a hyperlinked set of documents, such as the World Wide Web, with the purpose of measuring its relative importance within the set. It assigns relative scores to all nodes in the network based on the principle that connections to high-scoring nodes contribute more to the score of the node in question than equal connections to low-scoring nodes. Here, the description of the details of these social network metrics is out of the scope of this study. Readers who are interested in social network analysis can refer to [18] and [19].

III. DREX: A New Approach to Developer Recommendation

A. Problem Statement

To simplify, we assume that we have in sum l developers $dev_1, ..., dev_n$ in a software project and an open bug repository that contains m historical bug reports as $OBR = (br_1, ..., br_m)$ where br_i denotes a bug report. Following a bug report br_i, a set of comments $c_{br_i} = (c_{i,1}, ..., c_{i,|c_{br_i}|})$ were made correspondingly by a collection of developers $devs_{br_i} = (dev_{i,1}, ..., dev_{i,|c_{br_i}|})$ (here, $dev_{i,j}$ is an element of $\{dev_1, ..., dev_l\}$) who posted comments for the historical bug report and these comments have augmented the relevant information for resolving the bug. For easy reading of the paper, we would like to define the developer collection of a historical bug report as the following.

Definition 3.1: Developer collection of a historical bug report. For a historical bug report br_{his}, a collection of developers $devs_{br_i} = (dev_{i,1}, ..., dev_{i,|c_{br_i}|})$ posted comments for br_{his} in the open bug repository and we define this collection of developers as the developer collection of br_i. Note that one developer can occur many times in a developer collection.

The comments of a historical bug report are generated by either developers' manual appending or automatic appending by bug tracking system induced by developers' activities on this bug. The structure of a bug report and its followed comments is depicted in Figure 2. Because we do not know the real contribution of those developers to resolve the bug, we simply regard that all developers in $(dev_{i,1}, ..., dev_{i,|c_{br_i}|})$ who made comments for the historical bug report br_i have some expertise of resolving the bug. Thus, the developer recommendation problem can be stated as that, for a new bug report br_{new}, we need to find a model M that can produce a given number (Q) of developers $\{dev_{new,1}, ..., dev_{new,Q}\}$

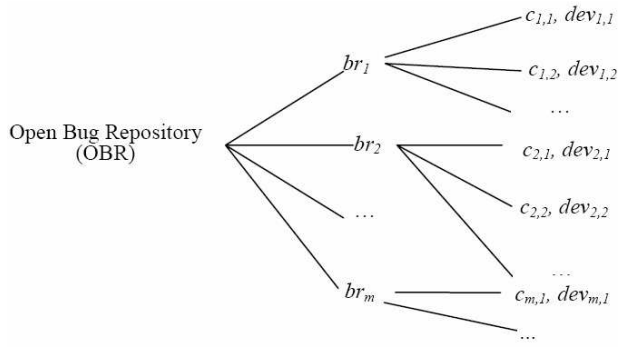

Fig. 2. The structure of open bug repository, bug reports, comments and developers.

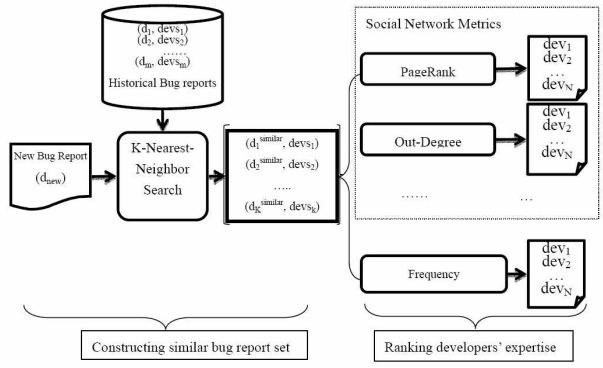

Fig. 3. The two components developed in DREX.

$(dev_{new,j}(1 \leq j \leq Q)$ is an element of $\{dev_1, ..., dev_l\}$ who have expertise in the new bug.

B. DREX: The Proposed Approach

When a meaningful bug report newly arrives at the open bug repository, our basic idea is to first search the similar historical bug reports of the new bug report, and then recommend the new bug report to developers who have enough expertise on the new bug report. Specifically, DREX comprises two components as shown in Figure 3. One component is to construct a similar bug report set of the new bug report based on similarities of the documents of historical bug reports and the document of the new bug report. The other component is to rank the developers expertise based on their participation records in the similar bug reports. Finally, the developers with high expertise on the similar historical bug reports are recommended to the new bug report. The following subsections are the details of these two components.

1) Constructing similar bug report set: To be consistent with the definition in Section II.B, a document for a historical bug report br_i could be written as $d_i = (br_i, c_{br_i})$ and document for a new bug should be written as $d_{new} = (br_i)$. As mentioned in Section II.A, each bug report consists of predefined fields and free-form textual content. In our approach, only free-form textual contents are used, that is, br_i denotes the textual content that merely includes the keywords and the description of the ith bug report. When a new bug report arrives, it is first transferred into a document vector $d_{new} = \{w_{new,1}, ..., w_{new,n}\}$ by document representation. Then, the documents of historical bug reports in the open bug repository are used to compute the similarities with the document of the new bug by Equation 2. Finally, the predefined number K historical bug reports, whose documents have top similarities with the document of the new bug report, are extracted from the open bug repository to construct the similar bug report set of the new bug report. That is, $SimSet(d_{new}, K) = \{d_1^{similar}, ..., d_K^{similar}\}$.

2) Ranking developers' expertise: After the similar bug set $Sim(d_{new}, K)$ of the new bug report br_{new} is constructed, we extract the corresponding developer collection $devs_{br_i}$ for each similar bug report d_i. Thus, the developers' participation records $(devs_{br_1}, ..., devs_{br_k})$ in similar bugs are embodied in their developer collections and are used to rank the developers' expertise on resolving the new bug. This study adopts seven ranking methods to rank the developers' expertise on the new bug. One is simple frequency of each developer's participation records in the comments and the other six metrics are adopted from social network analysis.

When using simple frequency for expertise ranking, for each developer dev_j, we count its occurrence in the developer collections of similar bug reports. With social work metrics, first, we construct a social network according to developers' participation records in the similar bug reports. For instance, assuming that the similar bug set of a new bug report represented as d_{new} contains two bug reports represented as d_1 and d_2, and correspondingly, their two developer sets are (dev_1, dev_2, dev_3) and (dev_1, dev_3) respectively, we establish the social network using the method as shown in Figure 4. Here, $sim(d_{new}, d_1)$ and $sim(d_{new}, d_2)$ denote the similarities of the new bug d_{new} and d_1 and d_2 which are computed with Equation 2. These similarities are used as the edge weights in the network. A developer in the latter places in a comment sequence points to all its previous developers (such as dev_3 pointing to both dev_1 and dev_2) because we hold that the developers in the latter position of a comment sequence using its pervious comments as references before they posted their comments. Second, we employed Indegree, Outdegree, Degree, PageRank, Betweenness and Closeness metrics to rank the developers' expertise on the new bug. Based on the above expertise ranking, Q developers who are ranked as having top expertise are recommended as potential contributors to resolving the new bug.

C. Evaluation

We adopted Precision and Recall to evaluate the performance of different methods on developer recommendation. They are widely used in information retrieval and text classification [13]. Equations 3 and 4 show how to compute Precision and Recall in the context of developer recommendation where $\{dev_{new,1}, ...dev_{new,Q}\}$ contains the recommended developers for the new bug report d_{new} and we use the real developers

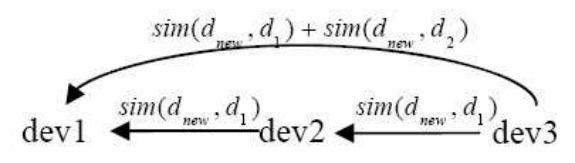

Fig. 4. The network constructed from the two developer sets (dev_1, dev_2, dev_3) and (dev_1, dev_3).

who posted comments to the testing bug reports as the ground truth in the following experiments.

$$Precision = \frac{|\{dev_{new,1}, ..., dev_{new,Q}\} \bigcap \{Ground\ Truth\}|}{|\{dev_{new,1}, ..., dev_{new,Q}\}|} \quad (3)$$

$$Recall = \frac{|\{dev_{new,1}, ..., dev_{new,Q}\} \bigcap \{Ground\ Truth\}|}{|\{Ground\ Truth\}|} \quad (4)$$

IV. Experiments

A. Experiment Setup

We use the data collected from the open bug repository of Mozilla Firefox to conduct the experiments in this study. Mozilla Firefox adopts Mozilla Bugzilla to manage its bug reports since 2002. In sum, about 77,996 Mozilla bug reports had been submitted to Bugzilla database from Feb 2002 to Aug 2009. We select 9,133 of these bug reports labeled with "fixed" and their comments from the Mozilla open bug repository to set up an initial bug data set and the following data cleaning procedures are imposed on these bug reports.

On one hand, we eliminate those bug reports whose size of developer collection is smaller than 3 because we find that the bug reports that have only 1 or 2 developers involved mostly are direct fixing (one developer found the bug and fixed it without any discussion with other developers).

On the other hand, we define parameter N as the number of bug report participation records of a developer in the open bug repository, and eliminate the developers whose N are smaller than 10 and larger than 1,000 from the developer collection of each bug report. We find that more than 5,322 developers participated in less than 10 bugs. In fact, most of these developers are software users who merely submitted bugs and never contributed to bug resolution. In Mozilla Firefox open bug repository, 4 developers participated in more than 1,000 bugs, and they are regarded as process people of Mozilla Firefox project such as project managers and QAs. Range of N for filtering some noise bugs is ad-hoc in this approach, but both project size and developers number are considered.

After data cleaning, we obtain 5,195 bug reports for experimentation in this study. Table 1 lists the number of bug reports with different sizes of developers. On average, each bug has 5.2 developers participating in its discussion. Table 2 shows the distribution of developers on the bug report participation.

TABLE I
THE NUMBER OF BUG REPORTS AND THEIR SIZES OF DEVELOPER COLLECTIONS.

# of bug reports	Size of developer collection
1,642	3
1,223	4
838	5
563	6
398	7
241	8
173	9
117	10

TABLE II
THE DISTRIBUTION OF DEVELOPERS OVER THE NUMBER OF BUG REPORT PARTICIPATION RECORDS.

N	# of developers
≥ 10	481
≥ 20	295
≥ 30	217
≥ 40	178
≥ 50	153
≥ 60	128
≥ 70	112
≥ 80	100
≥ 90	93
≥ 100	81

In order to evaluate DREX, we devise 10 cases under the conditions listed in Table 2 using the whole bug data set. For instance, when we set $N \geq 10$, the developers who participate in less then 10 bug reports are eliminated from the developer collections of all bug reports. In each case, we divide the data set into 5 folds and each fold has its training set and testing set determined by the submitted time of bug reports. For the space limitation, we do not list all the 10 cases and their folds in this paper. Table 3 shows the folds and their training and testing sets under the case with $N \geq 10$. Note that all the experiment results in the Section IV.B are averaged on the 5 folds when examining the performance of DREX.

B. Experiment Results

Two parameters, which are N and K, are needed to be tuned to examine the performances of DREX where K is clarified in Section III.B and N is clarified in Section IV.A. The parameter K is used to control the size of similar bug set and the parameter N is used to control the size of developer collections of historical bug reports. ML-KNN [14] is a multi-label k-nearest neighbor classifier, which is in particular useful when a real-world object is associated with multiple labels simultaneously. In the context of our experiment, each bug report is associated with multiple developers. Therefore, ML-KNN still follows the idea of text categorization approach. We compare DREX with the traditional text categorization in our experiment.

1) Tuning parameter K: In this experiment, we fix parameter N as 80, and tune parameter K from 10 to 25 as shown in Figure 5. We recommend 10 developers to each new bug in the testing set and calculate the fraction of real developers of the 10 developers, that is, Recall@10. PageRank, Degree,

TABLE III
THE FOLDS WITH THEIR TRAINING SET AND TESTING TEST UNDER CONDITION WITH $N \geq 10$. "BEGIN" MEANS THE BEGINNING TIME AND "END" MEANS THE END TIME OF THE SUBMITTED BUG REPORTS IN THE SETS.

Fold No.	# of training bugs ($begin \sim end$)	# of testing bugs ($begin \sim end$)
1	4894($2001-05-27 \sim 2009-02-02$)	50 ($2009-02-04 \sim 2009-03-02$)
2	4944($2001-05-27 \sim 2009-03-02$)	50 ($2009-03-02 \sim 2009-03-30$)
3	4994($2001-05-27 \sim 2009-03-29$)	50 ($2009-03-30 \sim 2009-04-25$)
4	5044($2001-05-27 \sim 2009-04-24$)	50 ($2009-04-25 \sim 2009-05-25$)
5	5094($2001-05-27 \sim 2009-05-25$)	50 ($2009-05-25 \sim 2009-06-21$)

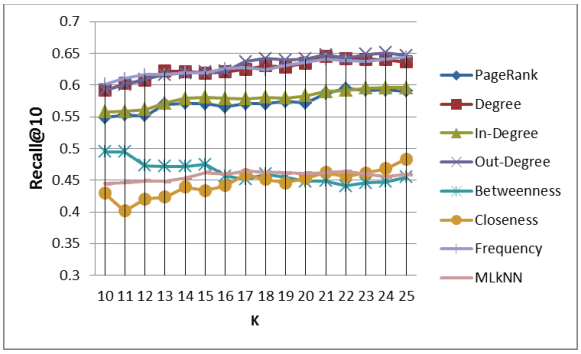

Fig. 5. Performances of DREX with different expertise ranking metrics in comparison with traditional multi-labled text classification when K is tuned from 10 to 25 with N is fixed as 80.

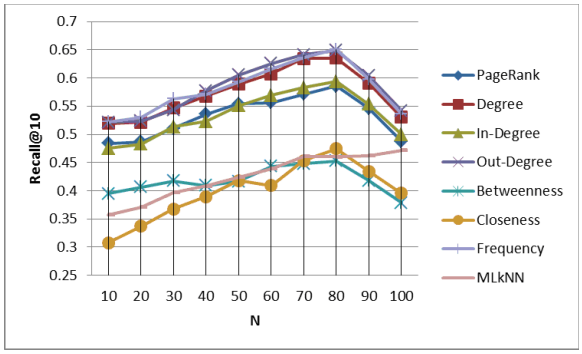

Fig. 6. Performances of DREX with different expertise ranking metrics in comparison with traditional multi-labled text classification when N is tuned from 10 to 100 when K is fixed as 20.

Indegree, Outdegree, Betweenness and Closeness metrics are adopted from social network analysis. Frequency metric count the number of comments a developer posted with the bug reports in the similar bug set. ML-KNN method involves a parameter K which has the same connotation as the K in DREX when it is used for multi-labeled classification [14].

We can see from Figure 5 that Frequency and Outdegree metrics outperform other metrics in developers' expertise ranking. With most metrics except Betweeness and Closeness, DREX has produced better performances than ML-KNN when K is varied from 10 to 25. This outcome illustrates that compared with other metrics, Frequency and Outdegree, which are closely relates with the number of comments a developers posted with the similar bugs, determines the developer's expertise. Moreover, each magnitude of performance variations of DREX embedded with different metrics is much smaller than that of ML-KNN. This outcome illustrates that DREX is more robust in developer recommendation than that originated from multi-labeled text classification.

2) Tuning parameter N: In this experiment, we fix the parameter K as 20, and tune parameter N from 10 to 100, with an interval of 10. Figure 6 shows the performances of DREX with different metrics compared with traditional method of multi-labeled text classification. It can be seen that DREX attains its best performance when N reaches 80. That is, DREX has produced its best performance under the constraint that each developer in the developer collection participated in at least 80 bug reports. However, for ML-KNN, its recall increases steady when N is tuned from 10 to 100. This outcome can be explained that when N becomes large, the size of each developer collection becomes small and the developers in social network become isolated nodes. However, for ML-KNN, this will not influence its working mechanism and the complexity of multi-label classification is decreased in huge granularity as pointed out in [14]. DREX and ML-KNN are more sensitive with the parameter N than those with another parameter K.

3) Recommending different number of developers: We fix N as 80 and K as 20, and compare the performance of DREX with different metrics and ML-KNN in recommending each new bug to different number of developers. Figure 7 and Figure 8 show the precision and recall of each method with different number(that is, Q in Section III.A) of developers recommended to new bug reports, respectively. We vary the number of recommended developers from 1 to 10.

It can be seen from Figure 7 and Figure 8 that the largest precision is derived with Frequency as nearly 0.70 when the top 1 developer is recommended and, the largest recall is derived also with frequency as nearly 0.65 when 10 developers are recommended. Expertise ranking with Betweenness and Closeness metrics cannot produce favorable performance. Nevertheless, DREX with any one of the other four social network metrics has produced better performance than ML-KNN in whatever precision and recall.

Because each bug has on average 5.2 developers in its developer collection, we roughly draw that with frequency metric, more than 3 developers are accurately recommended to each new bug when the number of recommended developers are set as 10. In contrast with the large number of developers (nearly 10,000) involved in the Mozilla Firefox project; we hold that

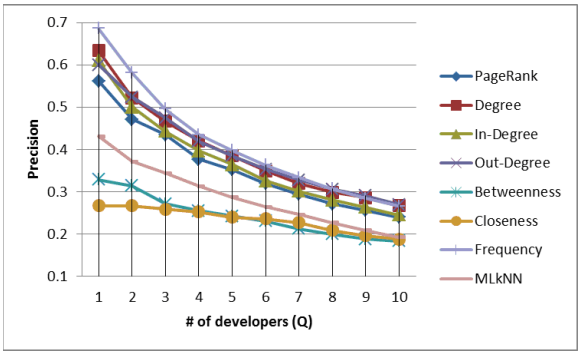

Fig. 7. Precisions of DREX with different metrics in compared with ML-KNN when recommending different number of developers to each new bug report.

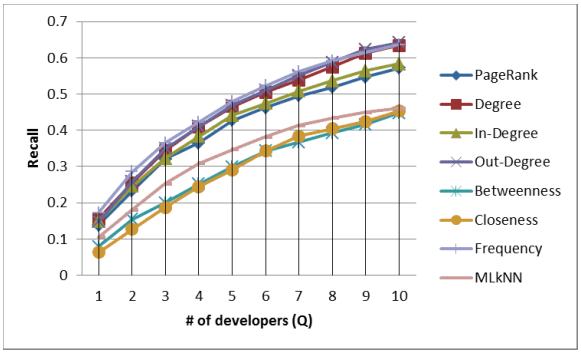

Fig. 8. Recalls of DREX with different metrics in compared with ML-KNN when recommending different number of developers to each new bug report.

this result is acceptable for real developer recommendation practice.

In most cases, Frequency performs slightly better than Out-degree metric. For instance, they both have 0.66 top-10 recall and 0.28 top-10 precision. With detailed inspection for the 250 testing bug reports, we find that in 200 testing bug reports, they produce the same developers for recommendation; Frequency outperforms Outdegree in 31 testing bug reports and Outdegree outperforms Frequency in 19 bugs. This outcome illustrates that Outdegree metric is an alternatives to simple Frequency in ranking developers' expertise.

V. Threats to Validity

There are several threats to validity of our study in this paper. First, we examine DREX using Mozilla Firefox project and we merely retain 250 bugs from 4th, Feb 2009 to 21st, June 2009 in Mozilla Firefox open bug repository for examining DREX. We hope Mozilla Firefox project are representative for most open source projects and those bug reports are representative for all Mozilla Firefox bug reports. Second, we use vector space model for document indexing and TF*IDF for document weighting with the purpose of transferring bug reports and their comments into numeric vectors. Thus, cosine similarity is employed to gauge the similarity of bug reports in KNN search. As mentioned in Section II, a bug report actually comprises many predefined meta-fields such as component_id, product_id, etc. besides freeform textual contents. In this study, we only used the freeform textual contents of a bug report to capture developers' expertise because we found that a group of developers who posted similar contents in bug discussion had same values in other meta-fields than freeform textual contents. We hope the processing will not bias our experiment and we will consider how to combine other meta-fields in recommending developers for a given bug report in our future sudty. Third, the developers' network was built using the developer collections of similar bugs. We do not adopt the method to link each pair of developers in a developer collection and consequently make a clique for a developer collection because we hold that the chronological information of the posted comments is useful to link developers. In our future work, we will attempt more network construction methods.

VI. Related Work

The related work of this study is mainly on automatic bug assignment. Many studies have been conducted with the goal of finding appropriate developers for resolving new bugs. These studies can roughly be divided into two main streams. One is to assign bugs to developers based on text categorization [5] [7] [8] [20] [21] and the other one is to model expertise of developers using historical to match bug contents [6] [10]. In the former stream, machine learning techniques such as naïve Bayes and support vector machines (SVMs) are used to categorize new bugs using historically assigned bug reports as training data. Essentially, the k-Nearest Neighbor component involved in DREX is much similar with text categorization, i.e., to categorize new bugs to the class predefined by its similar historical bugs. We also implement multi-labeled multi-class categorization using the collected Mozilla Firefox bug report data and the experimental results show that the performances of text categorization are not comparable to that of DREX in this study. In the latter stream, historical bug resolving records [10] and change history in source code repository [6] are used to model the expertise of developers. Then, new bugs and developers' expertise are matched to recommend bugs to the developers with expertise of those bugs. In our study, we may regard the developers belonging to the similar bug set have the expertise relevant with the new bug. Thus, different metrics are adopted to rank developers' expertise on the new bug.

Among the methods for bug assignment mentioned above, the most related work to ours was conducted by J.Anvik [21]. He proposed a machine learning based triage-assisting recommender creation framework that includes three parts. The first parts is to recommend developers to whom a given bug report should be assigned. The second part is to relate bug reports with product components. The third part is to recommend developers of carbon copy list when a bug report is submitted. Nevertheless, it should be noted that DREX is different from his work. First, we maintain that bug fixing is a social process and a group of developers have contributed

their ideas and advices in the process of fixing the bug. Thus, DREX introduces social network analysis for recommending developers for bug resolution. Second, our approach regards developers who posted comments followed with bug reports as the ground truth of interested developers, not developers in carbon copy list. We noticed that although some developers are in the carbon copy list of a bug report, but they never contribute to bug resolution.

Another related work to this study is on social communities of OSS projects, such as the email social network[19], stakeholder network [18] and the social structure of OSS project [22][23]. The social network analysis introduced in this study has different purpose with theirs.

VII. Conclusion

This work proposed a new approach called DREX to developer recommendation based on KNN search and expertise ranking for resolving bugs. DREX utilizes the bug reports and their comments in open bug repository to recommend developers to the bug reports in which those developer have the resolution potential. When a new bug report was submitted to the open bug repository, DREX first searches for similar historical bug reports of the new bug and the developers who contributed to the solution of these historical bug reports are retrieved. Second, DREX adopts various metrics to rank the expertise of these retrieved developers according to their participation records in these historical bugs.

Our experimental results on Mozilla Firefox open bug repository demonstrated that DREX outperforms the traditional bug assignment method which is based on multi-labeled text classification. We also investigated various metrics including simple Frequency and six social network metrics on expertise ranking of DREX. The experimental results demonstrated that, simple Frequency and Outdegree perform the best in recommending new bug reports to potential developers and, both of their recalls attain up to 0.6 when recommending each new bug to top 10 developers. All these results have shown the promising aspects of DREX.

The future work of this study will examine DREX on more open bug repositories for a comprehensive comparison with other automatic bug assignment methods. More methods of network construction will be investigated to utilize the social communities inherent in software projects. Besides the free-form textual contents of bug reports, our future work will take predefined meta-fields of bug reports into consideration especially "component" field when gauging bug report similarities.

Acknowledgments

We appreciate the help of Mr. Guy Pyrzak for providing us with the Mozilla Firefox open bug repository. This work is supported by the National Natural Science Foundation of China under Grant Nos. 60873072, 61073044, 60903050 and 71101138; the National Science and Technology Major Project; the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry.

References

[1] J. Anvik, L. Hiew, and G. C. Murphy, "Coping with an open bug repository," in *Prooceedings of Eclipse'05*, 2005.

[2] W. Wu, W. Zhang, Y. Yang, and Q. Wang, "Time series analysis for bug number prediction," in *Proceedings of Second International Conference on Software Engineering and Data Mining*, pp. 589–596, 2010.

[3] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, ""not my bug!" and other reasons for software bug report reassignments," in *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, CSCW '11, pp. 395–404, 2011.

[4] D. Bertram, A. Voida, S. Greenberg, and R. Walker, "Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams," in *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, CSCW '10, pp. 291–300, 2010.

[5] D. Cubranic and G. C. Murphy, "Automatic bug triage using text categorization," in *Proceedings of the 16th International Conference on Software Engineering & Knowledge Engineering*, pp. 92–97, 2004.

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

[7] J. Anvik, "Automating bug report assignment," in *Proceedings of the 28th International Conference on Software Engineering*, pp. 937–940, 2006.

[8] 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, 2006.

[9] 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, 2007.

[10] J. Anvik and G. C. Murphy, "Determining implementation expertise from bug report," in *Proceedings of the 4th International Workshop on Mining Software Repositories*, pp. 1–8, 2007.

[11] G. Salton, A. Wang, and C. S. Yang, "A vector space model for information retrieval," *Journal of the American Society for Information Science*, vol. 18, pp. 613–620, 1975.

[12] M. Gegick, P. Rotella, and T. Xie, "Identifying security bug reports via text mining: An industrial case study," in *Proceedings of the 7th Working Conference on Mining Software Repositories*, pp. 11–20, 2010.

[13] W. Zhang, T. Yoshida, and X. Tang, "A comparative study of tf*idf, lsi and multi-words for text classification," *Expert Systems with Applications*, vol. 38, no. 3, pp. 2758–2765, 2011.

[14] M. Zhang and Z. Zhou, "Ml-knn: A lazy learning approach to multi-label learning," *Pattern Recognition*, vol. 40, no. 7, pp. 2038–2048, 2007.

[15] S. Wasserman and K. Faust, *Social Networks Analysis: Methods and Applications*. Cambridge, United Kingdom: Cambridge University Press, 1994.

[16] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of Mathematical Sociology*, vol. 25, no. 2, pp. 163–177, 2001.

[17] 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*, ESEC/FSE '09, pp. 111–120, 2009.

[18] S. L. Lim, D. Quercia, and A. Finkelstein, "Stakenet: Using social networks to analyse the stakeholders of large-scale software projects," in *Proceedings of 32nd International Conference on Software Engineering*, pp. 295–304, 2010.

[19] C. Bird, A. Gourley, P. Devanbu, M. Gert, and A. Swaminathan, "Mining email social networks," in *Proceedings of the 3rd International Workshop on Mining Software Repositories*, p. 137C143, 2006.

[20] J. Xuan, H. Jiang, Z. Ren, J. Yan, and Z. Luo, "Automatic bug triage using semi-supervised text classification," in *Proceedings of 22nd International Conference on Software Engineering and Knowledge Engineering*, pp. 209–214, 2010.

[21] J.Anvik, *Assisting bug report triage through recommendation*. PhD thesis, University of British Columbia.

[22] C. Bird, D. Pattison, R. DSouza, V. Filkov, and P. Devanbu, "Latent social structure in open source projects," in *Proceedings of SIGSOFT 2008/FSE*, p. 16, 2008.

[23] K. Crowston and J. Howison, "The social structure of free and open source software development," *First Mondy*, vol. 10, no. 2, pp. 189–195, 2005.