View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by espace@Curtin

Copyright © 2014 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Duplicate Bug Report Detection Using Clustering

Raj P. Gopalan and Aneesh Krishna
Department of Computing
Curtin University
Perth, Western Australia
{R.Gopalan, A.Krishna}@curtin.edu.au

Abstract— **Bug reporting and fixing the reported bugs play a critical part in the development and maintenance of software systems. The software developers and end users can collaborate in this process to improve the reliability of software systems. Various end users report the defects they have found in the software and how these bugs affect them. However, the same defect may be reported independently by several users leading to a significant number of duplicate bug reports. There are a number of existing methods for detecting duplicate bug reports, but the best results so far account for only 24% of actual duplicates. In this paper, we propose a new method based on clustering to identify a larger proportion of duplicate bug reports while keeping the false positives of misidentified non-duplicates low. The proposed approach is experimentally evaluated on a large sample of bug reports from three public domain data sets. The results show that this approach achieves better performance in terms of a harmonic measure that combines true positive and true negative rates when compared to the existing methods.**

***Keywords-** bug report, duplicate detection, clustering, Bugzilla*

I. INTRODUCTION

As software development becomes increasingly complex, there is also greater pressure to release the products quickly. This often leads to software with many defects being made available to users. Software defects have caused loss of income running into many billions of dollars [17]. Fixing software defects is one of the most frequent software maintenance activities which amounted to 70 billion dollars in US alone [1]. Defect reporting is a vital part of the software development, testing and maintenance process. The purpose of a defect report is to state the problem as clearly as possible so that developers can replicate the defect easily and fix it. A bug tracking system is designed to help developers keep track of reported bugs in their software products [6], [8], [11], [13]. A reporting system like Bugzilla allows users to report the bugs they encounter in software such as Mozilla and Eclipse [15]. By promptly processing these bug reports, the reliability of the software can be improved quickly. When several users submit bug reports for the same problem, these reports are called duplicate bug reports. If an incoming bug report describes a defect not seen before, then it should be assigned to the developers for fixing the bug. However, if it is a duplicate, it can be attached to the corresponding original master bug report. This process is referred to as triaging [1].

TABLE I. A BUG REPORT FROM MOZILLA BUG TRACKING SYSTEM

Bug Id	259814
Summary	Find highlighting disables point-in-text clicking in input fields
Description	If you find text, have the "Hilight" button turned on, and that text appears within an input field, you cannot click to place the input cursor in the middle of the found text.
Product	Mozilla Toolkit
Component	Find Toolbar
Version	Trunk

Bug reports usually include details such as a bug id, a summary and a description. It may also contain other details such as product, component and version. The fields used for different projects can vary to some extent, though they tend to be similar in content. Table I shows a sample bug report with values for different fields from the Mozilla bug tracking system.

A bug reporting system for a widely used software product may receive a large number of bug reports, especially after new releases of the software. Usually, bug triagers need to manually go through the list of bug reports to determine if they are duplicates or not. This consumes a large amount of time and effort by the triagers. For example, almost 300 bugs a day were reported for Mozilla in 2005 that overwhelmed the capacity of the programmers available [1]. To mitigate such large demands on triaging and fixing of defects, there is need for an automated tool that can assist in determining if a bug report is a duplicate.

So far two main approaches have been proposed for detecting duplicate bug reports. The first approach attempts to directly identify duplicates to prevent them from reaching triagers [5]. The second approach provides a list of top-k most similar bug reports for each new report from which a triager needs to identify whether it is a real duplicate [3], [5], [12]. As some bug reports may not contain sufficient information for fixing the defect, the top-k similar reports can often help to provide the missing details. However, the accuracy of the existing techniques for duplicate bug report detection is still relatively low [1], [4], [5].

In an early study of duplicate bug report detection, Runeson et al [4] proposed the use of natural language processing techniques to rank similar bug reports. However, they relied only on the textual information without regard to

the other features available in Bugzilla such as component where the bug resides, product version, report priority, etc. The accuracy of this approach was relatively low. In addition to the natural language processing techniques, Wang et al [3] used execution traces to detect defects. Execution traces are hard to get for specific bugs and often unavailable for typical bug reports. Sureka and Jalote [2] proposed a character N-gram based model for duplicate bug detection. They evaluated the method using top-N similar reports on a random sample of 1100 bug reports. However, their results were not compared with existing approaches in the literature.

Jalbert and Weimer [5] proposed a method that uses surface features, textual semantics and graph clustering to predict duplicate status of bug reports. They were able to filter up to 8% of duplicates while allowing every real defect to reach the developers. Sun et al [1] developed a discriminative model for bug report retrieval using a support vector machine (SVM). The reports were modeled as bags of words and the similarity between a pair of reports was computed as the sum of the inverse document frequencies for the words in common. The method is based on identifying the top-k similar reports from which the most probable duplicate pair is determined using SVM. In [7], Sun et al extended BM25F to a new similarity measure which is a linear combination of textual and categorical features. Tian et al [6] improved the methods described in [1], [7] by refining the similarity measure in [7] and used SVM to identify the duplicates. They improved the identification of duplicates to 24% compared to only 8% in [5]. However, the true negative rate declined by 9% compared to [5].

In this paper, we investigate whether a clustering based approach can improve the accuracy of duplicate detection. The proposed method uses the textual information contained in the summary and description fields of the bug reports. Pairs of bug reports are compared using Cosine similarity. A threshold is specified to determine whether the similarity between two bug reports is significant enough for them to be clustered together. A new bug report is compared with the cluster representatives of existing clusters and will be added to the cluster with which it is most similar. If its similarity is below the threshold for all clusters, it is added to a new cluster. We have evaluated our approach on three large data sets of bug reports from Mozilla, Eclipse and Open Office projects. The results are compared against the best performing approach from the literature.

The previous work closest to our approach is that of Jalbert and Weimer [5] who also used cosine similarity to compare pairs of bug reports along with a graph clustering algorithm to detect duplicates. We also use cosine similarity in this paper, though other similarity measures such as Jaccard coefficient and Pearson correlation coefficient [18] could be substituted in our clustering algorithm. Instead of the graph clustering approach used in [5], but we adapt the INCLUS algorithm which was originally proposed for clustering high dimensional sparse transactional data [18]. Tian et al. [6] followed a similar approach to [5] in formulating the evaluation measures for their duplicate bug report detection method and therefore could compare their results with that of [5]. In this paper, we compare our results with those in both [5] and [6].

The remaining sections of this paper are organized as follows: Section 2 presents our approach for duplicate bug report detection. Section 3 describes the experiments, results and evaluation of the approach. We conclude the paper and present the future directions in Section 4.

II. Proposed Approach to Detecting Duplicate Bug Reports

Bug reports normally include free form textual descriptions and titles and duplicate bug reports may share the same words. A bug report always contains two important fields: summary and description along with its bug id as shown in Table I. It may also contain other features such as product, component, version and priority. Similarity between two bug reports may be considered based on the summary field alone, the description field alone or the union of these two. Besides textual similarity, we may also consider product, component, version and priority fields in determining whether two bug reports are duplicates.

A. Document Similarity Measure

In this paper, we consider the document similarity between two bug reports only within the same corpus. A corpus may consist of bug reports of a particular project such as Mozilla, Open Office or Eclipse. In our experiments, the summary and description fields of the bug report are considered as part of one corpus. We adapt the cosine similarity measure presented in [5] for comparing two bug reports. We consider the set of n unique words that are present in the entire corpus. Each bug report in the corpus is represented by a vector v of size n, where $v[i]$ is the number of occurrences of the word i in that report.

For vectors v1 and v2 that represent two bug reports, the cosine similarity is computed using Equation (1) where v1·v2 represents the dot product:

$$similarity = cos(\theta) = (v1.v2)/(v1 \times v2) \qquad (1)$$

The smaller the value of θ, the two vectors are closer to being collinear, and hence more weighted words are shared by the two reports.

B. Duplicate Detection

Clustering is used in our approach to group bug reports that are similar based on the cosine similarity measure. Two bug reports that share a high proportion of common words in their summary and description fields have a greater probability of being duplicates. The input bug reports are treated as if they are separate data streams based on distinct values of product ID and component ID. Only bug reports of

the same product and component are clustered together. Bug reports that share few common words could also describe the same defect, but the proposed approach based on the bag of words representation is not suitable for detecting such duplicates.

The bug reports in the repository are organized in an ascending order of bug ID's. As bug reports are consecutively numbered, the duplicate reports will always follow the first report of a given defect. Each report is represented as a set of fields that includes summary and description containing a set of words and their frequencies of occurrence. For a given pair of reports, cosine similarity is computed separately for the summary and description fields. These values are then combined by applying a suitable weighting w in the range 0-1 chosen by experimentation. The weighted similarity of a pair of bug reports is computed as $(sw + d(1-w))$, where s and d are the similarity values for the summary and description fields. If the weighted similarity is above a threshold, the two reports are considered to be potentially duplicates. The choice of threshold and the relative weighting of summary and description fields are determined experimentally by clustering a sample of the bug reports in a given repository in which the actual duplicates have been previously identified. These parameters are chosen such that the clustering results match most closely with the actual.

C. *Clustering Algorithm*

The Clustering algorithm we use is an adaptation of the INCLUS algorithm proposed in [18]. Unlike INCLUS, our algorithm uses the first report in a cluster as its representative and requires as the input parameters a similarity threshold and a weighting factor to be applied to the similarity of summary and description fields. INCLUS uses a set of frequent items in each cluster as the representative with which to compare new transactions and requires a support threshold and a similarity threshold as clustering parameters.

Algorithm : Clustering of duplicate bug reports

Input : A set of bug reports, a similarity threshold
Output: Clusters of master reports and their duplicates.

1. Insert the first bug report into a new cluster and nominate it as the cluster representative.

2. While NOT end of bug reports
   - **a.** Read the next bug report.
   - **b.** For each existing cluster, do
     - **i.** Compute Document similarity between the current report and the cluster representative.
     - **ii.** If current similarity value > the max similarity of current report with any cluster, store the cluster number and the current similarity value.
   - **c.** If max similarity >= threshold for current report, then Add it to the corresponding cluster else insert it into a new cluster.

Figure 1. Algorithm for Clustering bug reports

The algorithm is described in Fig. 1. It begins by inserting the first report into a new cluster. The first report of each cluster is treated as the cluster representative. Starting with the second report in the repository, the similarity of each report with the cluster representatives of existing clusters is computed as a weighted sum of the cosine similarity of the summary and description fields. If the maximum of the similarity values for the report with the cluster representatives is above the given threshold, it is inserted into the cluster with which it has the highest similarity. If the maximum similarity is below the threshold, it is inserted into a new cluster and designated as its cluster representative.

III. EVALUATION

This section describes the evaluation measures used, the setup of the experiments and the results obtained. We compare our results with the best results reported previously.

A. *Evaluation Measures*

As in [6], we adopt the approach of Jalbert and Weimer [5] to identify true and false positives, and also the definitions of true positive rate and true negative rate. The set of true positives denoted by *TP* consist of of duplicate bug reports correctly identified by the proposed approach. The set of false positives denoted by *FP* consist of bug reports that are incorrectly identified as duplicates. *TPRate* is defined as the ratio of the cardinality of *TP* to that of the actual set of duplicates denoted as *ActualDuplicates*.

$$\text{TPRate} = |\text{TP}| / |\text{ActualDuplicates}| \quad (2)$$

Similarly, TNRate is defined as the ratio of the number of non duplicate bug reports less the number of false positives as identified by our algorithm to the actual number of non duplicates denoted as *NonDupliactes*.

$$\text{TNRate} = |\text{NonDuplicates} - \text{FP}| / |\text{NonDuplicates}| \quad (3)$$

In order to balance the tradeoff between the true positive rate and the true negative rate, we also use the measure of Harmonic mean defined in [6] as

$$\text{Harmonic} = (2*\text{TPRate}*\text{TNRate}) / (\text{TPRate} + \text{TNRate}) \quad (4)$$

B. *Experimental Results*

We have evaluated our approach using the bug repositories of the large open source projects of Mozilla, Eclipse and Open Office that are available in the public domain for research and experimental purposes. These projects have been commonly used in the literature for evaluation of duplicate bug detection methods [1], [2], [3], [5], [6]. The details of the date ranges and the number of bug reports in each data set are given in Table II.

TABLE II. DETAILS OF DATA SETS USED IN THE EXPERIMENTS

Data set name	Period	Number of reports
Eclipse	Jan-Dec 2008	45234
Mozilla	Jan-Dec 2010	75653
Open office	Jan-Dec 2010	31138

A prototype of our approach was implemented in Java and experiments were carried out using Eclipse on an Intel Core i7 PC with 8GB memory running Windows 8. The bug reports were clustered using the algorithm described in Section 3. Initially, we used samples of bug reports from the three datasets containing both duplicate and non-duplicate bug reports, to determine the parameters of similarity threshold and the relative weighting of summary and description fields in the similarity calculation. The sample sizes used were about 16% of each data set. The parameters were chosen such that the TPRate on the sample is above 24% and the TNRate is close to 90% with a Harmonic value of above 39%. These values were used as they correspond to the previous best results reported in [6]. The parameters values so chosen are given in Table VI, which vary for the different data sets. These parameters were then applied to separate test data of similar size as the training data. Tables III, IV and V show the experimental results using the test data. These experiments covered the threshold values for similarity from 0.1 to 0.8 for each of the three data sets for further comparisons and discussion. Most of the existing research papers with the exception of [5] and [6] report only recall rates which are equivalent to the TPRate and do not report the TNRate. So the range of TPRate values we report can provide a basis for comparison with these papers.

TABLE III. EXPERIMENTAL RESULTS FOR ECLIPSE 2008 BUG REPORTS

Threshold	TPRate	TNRate	Harmonic
0.1	0.69	0.22	0.34
0.2	0.64	0.44	0.52
0.3	0.58	0.64	0.61
0.4	0.51	0.74	0.6
0.5	0.41	0.85	0.56
0.6	0.33	0.88	0.48
0.7	0.28	0.93	0.43
0.8	0.25	0.96	0.39

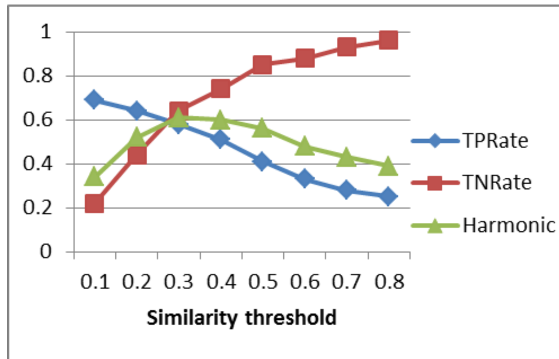

Figure 2. TPRate, TNRate, and Harmonic for Eclipse 2008 at different similarity thresholds

The values of similarity threshold, TPRate, TNRate, and Harmonic for the bug reports of Eclipse in 2008 are shown in Table III. The graph in Fig. 2 shows the TPRate, TNRate and the Harmonic plotted against the threshold values of similarity. Similarly, the experimental results for data sets from Mozilla and Open Office are shown in Tables V and VI, and also in Fig. 3 and 4 respectively.

From Table III and Fig. 2, it can be observed that as the similarity threshold value increases, the TPRate starting with a value of 0.69 for the threshold of 0.1 decreases continuously. TNRate, on the other hand, starts with a low value and increases to nearly 1. The Harmonic value increases with the threshold value up to a threshold of 0.3, and then gradually decreases for higher thresholds. Ideally, we would prefer the TNRate to be close to 1 and the TPRate to be as high as possible, so that the non-duplicate reports that need to be fixed are not missed while maintaining a high detection rate for duplicate bug reports. At the highest Harmonic value of 61% for Eclipse the TPRate is 58% and the TNRate 64%. As we require the TNRate to be closer to 1, the TPRate of 33% corresponding to detection of 33% of actual duplicates with TNRate of 88% which represents a false positive rate of 12% are more viable. The TPRate is significantly better than the best value of 24.48% reported by existing methods in the literature, though our TNRate is lower than in that study by 3% [6]. However our Harmonic value of 48% is much better than 39% achieved in the same study.

TABLE IV. EXPERIMENTAL RESULTS FOR MOZILLA 2010 BUG REPORTS

Threshold	TPRate	TNRate	Harmonic
0.1	0.45	0.2	0.27
0.2	0.39	0.43	0.41
0.3	0.32	0.68	0.44
0.4	0.27	0.86	0.41
0.5	0.13	0.92	0.23
0.6	0.07	0.99	0.12
0.7	0.02	0.99	0.03
0.8	0.02	0.99	0.03

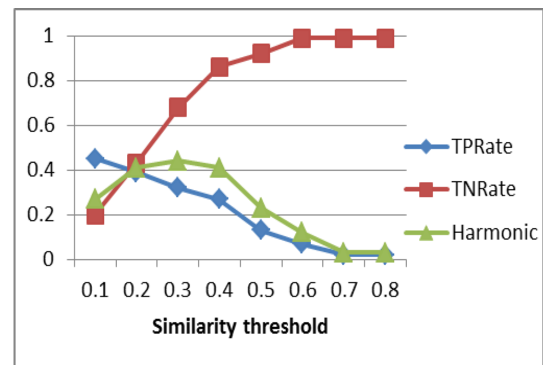

Figure 3. TPRate, TNRate, and Harmonic for Mozilla 2010 at different similarity thresholds

The experimental results for the Mozilla data set are shown in Table IV and Fig. 3. The pattern of TPRate, TNRate and Harmonic for different threshold values is similar to that for the Eclipse data. However, the absolute values of these measures are significantly better for the Eclipse data at corresponding threshold values. The TPRate of 27% at a threshold value of 0.4 is better than the previously reported best value of 24% for Mozilla bug reports [6]. So also is our Harmonic value of 41% compared to the previous best result of 39%, though the TNRate is lower by 10% compared to that study. It is seen from the graphs in Figures 2-4, that there is a clear tradeoff between higher TPRates and lower TNRates as they move in opposite directions for varying threshold values.

TABLE V. EXPERIMENTAL RESULTS FOR OPEN OFFICE 2010 BUG REPORTS

Threshold	TPRate	TNRate	Harmonic
0.1	0.41	0.31	0.35
0.2	0.43	0.52	0.47
0.3	0.35	0.71	0.47
0.4	0.28	0.85	0.42
0.5	0.19	0.91	0.31
0.6	0.13	0.97	0.23
0.7	0.09	0.98	0.16
0.8	0.08	0.98	0.14

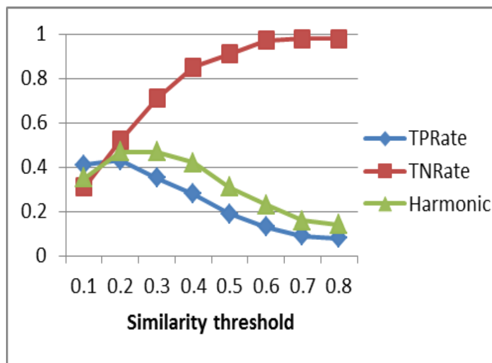

Figure 4. TPRate, TNRate, and Harmonic for Open Office 2010 at different similarity thresholds

Table V and Fig. 4 show the experimental results for the Open Office data set. The TPRate of 28% and a TNRate of 85% at the threshold value of 0.4 is comparable to the results for Mozilla data in Table IV. For the higher TNRate of 91%, the TPRate is lower at 19%. The Harmonic value peaks at 0.47 for a threshold of 0.3 and then gradually decreases. From the results for all three data sets, our current similarity measure and the parameter settings work best for the Eclipse data, though the results for the two data sets are comparable to the best results previously reported in the literature [6]. There is scope for improving our results further by choosing different similarity measures and fine tuning the parameter settings to suit different data sets.

C. *Discussion*

The results for the test data sets using our approach are compared against the results in [5] and [6] as only these two papers have previously reported both the TPRate and the TNRate. Other publications on the detection of duplicate bug reports evaluate their approaches based on the TPRate without accounting for the false positives [1], [2], [3], [4], [7]. If the corresponding false positives and the low TNRate are not considered, it is possible to get relatively high TPRate.

TABLE VI. CLUSTERING PARAMETERS CHOSEN BY EXPERIMENTS

Data set	Similarity threshold	Similarity weighting
Mozilla	0.4	0.9
Eclipse	0.7	0.4
Open Office	0.4	0.7

TABLE VII. COMPARISONS WITH PREVIOUS RESULTS

Approach	Data set	TPRate	TNRate	Harmonic
Jalbert and Weimer	Mozilla	8%	100%	15%
Tian, Sun and Lo	Mozilla	24%	91%	39%
Proposed in this paper	Mozilla	27%	86%	41%
	Open Office	28%	85%	42%
	Eclipse	33%	88%	48%

A comparison of the results for the three data sets using our approach with the previous results in [5] and [6] are shown in Table VII. References [5] and [6] have reported only on Mozilla data set. In making this comparison, we are assuming that the underlying characteristics of duplicate bug reports for various projects and in particular for the Mozilla project are similar over the life of the project. The TPRate and the Harmonic are higher for all three data sets when using our method. The TNRate on the otherhand is lower. As the Harmonic combines the TPRate and the TNRate into a single measure, it can be concluded that our method performs better than the previous approaches. Several researchers have previously evaluated their methods using the recall rate defined as the ratio of the number of correctly retrieved duplicates divided by the total number of actual duplicates. Most of these methods also retrieve for a given bug report the top-k similar bug reports of which it may be a potential duplicate. The value of k may vary from 1 to 20. The lowest recall rates in these studies are reported when the value of k is 1. For evaluation against these studies, the highest TPRate from our method can be treated as the top-1 retrieval. Based on this comparison, the top-1 recall rate of our approach is also higher than the top-1 recall rates of previous methods for the three data sets of Eclipse, Mozilla and Open Office projects as reported in [19].

IV. CONCLUSIONS

The need for automating the duplicate bug detection process is well recognized in the literature. In this paper, we extend the previous work by Jalbert and Weimer (2008) and Tian et al (2012), by proposing a clustering based approach to improve the accuracy of duplicate detection. The textual information in the summary and description fields of the bug reports were used for this task. Pairs of bug reports were compared using Cosine similarity with a threshold to determine whether they should be in the same cluster. We have evaluated our approach using three large data sets from Mozilla, Eclipse and Open Office projects. The results were compared against the previous approaches that have reported both true positive and negative rates for duplicate detection. For Mozilla and Open Office data sets, our results are better on TPRate and the overall Harmonic value, but slightly lower on the TNRate. Our results are significantly better for Eclipse data on TPRate, TNRate and Harmonic. However, the comparable previous studies had reported results based only on the Mozilla dataset (Jalbert and Weimer, 2008, Tian et al., 2012). Our experimental results indicate that the proposed approach presents a trade-off between high levels of duplicate detection and low levels of false positives.

The bag of words approach followed in this research cannot correctly deal with duplicate reports that share few common words. As future work, we plan to extend this approach by considering synonyms and phrases of similar meaning in comparing reports. It is also proposed to release a tool that will help developers flag duplicate bug reports as part of bug management in systems such as Bugzilla.

ACKNOWLEDGMENT

The authors wish to thank Rohit Gopalan (Visagio Australia) for his contributions towards the success of this project.

REFERENCES

[1] C. Sun, D.Lo, X.Wang, J.Jiang, S.-C.Khoo, “A discriminative model approach for accurate duplicate bug report retrieval,” Proc. International Conference on Software Engineering, pp. 45-56, 2010.

[2] A.Sureka, P.Jalote,“Detecting duplicate bug report using character n-gram-based features,” Proc. Asia Pacific Software Engineering Conference, pp. 366-374, 2010.

[3] X.Wang, L.Zhang, T.Xie, J.Anvik and J.Sun,“An approach to detecting duplicate bug reports using natural language and execution information,” Proc. International Conference on Software Engineering, pp. 461-470, 2008.

[4] P.Runeson, M.Alexandersson and O.Nyholm,“Detection of duplicate defect reports using natural language processing,” Proc. International Conference on Software Enginnering, pp. 499-510, 2007.

[5] N.Jalbert and W.Weimer, “Automated duplicate detection for bug tracking systems,” Proc. International Conference on DSN, pp. 52-61, 2008.

[6] Y.Tian, C.Sun, and D.Lo, “Improved duplicate bug report identification,” Proc. European Conference on Software Maintenance and Reengineering (CSMR), pp. 385-390, 2012.

[7] C. Sun, D.Lo, S.-C.Khoo, and J.Jiang,“Towards more accurate retrieval of duplicate bug reports,” Proc. International Conference on Automated Software Engineering (ASE), pp. 253-262, 2011.

[8] N.Bettenburg, R.Premraj, T.Zimmermann, and S.Kim, “Extracting structural information from bug reports,” Proc. International Working Conference on Mining Software Repositories,(MSR’08), pp. 27-30, 2008.

[9] J.Anvik, L.Hiew, and G.C.Murphy, “Copying with an open bug repository,” Proc. OOPSLA Workshop on Eclipse Technology eXchange (Eclipse’05), pp. 35-39, 2005.

[10] N.Bettenburg, R.Premraj, T.Zimmermann, and T.Sunghun Kim, “Duplicate bug reports considered harmful … really?,” Proc. IEEE International Conference on Software Maintenance (ICSM 2008), pp. 337-345, 2008.

[11] N.Bettenburg, R.Premraj, S.Just, A.Schroter, C.Weiss, and T.Zimmermann, “What makes a good bug report?,” IEEE Transactions on Software Engineering, vol.36, pp. 618-643, 2010.

[12] H.Cheng, X.Yan, J.Han, and C.-W.hsu, “Discriminative frequent pattern analysis for effective classification,” Proc. ICDE 2007.

[13] A.Podgurski, D.Leon, P.Francis, W.Masri, M.Minch, JB.Wang, “Automated support for classify software failure reports,” Proc. ICSE 2003.

[14] http://www.mysmu.edu/faculty/davidlo/

[15] http://www.bugzilla.org/

[16] https://issues.apache.org/ooo/

[17] G.Tassey, “The economic impacts of inadequate infrastructure for software testing”, National Institute of Standards and Technology - Planning Report 02-3.2002, 2002.

[18] Y.Li and R.P.Gopalan, “Clustering high dimensional sparse transactional data with constraints”, Proc. IEEE International conference on Granular Computing, 2006, pp. 692-695.

[19] A.T.Nguyen, T.T. Nguyen, T.N. Nguyen, D.Lo, and C.Sun, “Duplicate bug report detection with a combination of information retrieval and topic modeling”, Proc. ASE’12, Germany, pp. 70-79, September 2012.