Improved Duplicate Bug Report Identification

Yuan Tian[1], Chengnian Sun[2], and David Lo[1]
[1]*School of Information Systems, Singapore Management University*
[2]*School of Computing, National University of Singapore*
Email: yuan.tian.2011@exchange.smu.edu.sg, suncn@nus.edu.sg, davidlo@smu.edu.sg

***Abstract*—Bugs are prevalent in software systems. To improve the reliability of software systems, developers often allow end users to provide feedback on bugs that they encounter. Users could perform this by sending a bug report in a bug report management system like Bugzilla. This process however is uncoordinated and distributed, which means that many users could submit bug reports reporting the same problem. These are referred to as duplicate bug reports. The existence of many duplicate bug reports may cause much unnecessary manual efforts as often a triager would need to manually tag bug reports as being duplicates. Recently, there have been a number of studies that investigate duplicate bug report problem which in effect answer the following question: given a new bug report, retrieve k other similar bug reports. This, however, still requires substantive manual effort which could be reduced further. Jalbert and Weimer are the first to introduce the direct detection of duplicate bug reports; it answers the question: given a new bug report, classify if it as a duplicate bug report or not. In this paper, we extend Jalbert and Weimer's work by improving the accuracy of automated duplicate bug report identification. We experiments with bug reports from Mozilla bug tracking system which were reported between February 2005 to October 2005, and find that we could improve the accuracy of the previous approach by about 160%.**

***Keywords*-Duplicate bug reports; Relative similarity; Bugzilla**

I. Introduction

Bugs are prevalent; software defects are found in most if not all software systems. Bug reporting process is a way to elicit feedback from end users on defects and failures that affect them. Systems like Bugzilla or Jira are frequently used to aid this bug reporting process. From the software developers side, special personnel often referred to as bug triagers, would then go through these bug reports and assign suitable developers to address a particular report. Not all reports are assigned, some reports are dropped after screening by the bug triagers.

Despite the benefit of bug reporting process, bug reporting in nature is an uncoordinated distributed process. The same defect and failure could affect many users. These users could simultaneously or in parallel submit reports describing the same defect. These reports are termed as *duplicate bug reports*. Bug triagers should not assign these reports to different developers; this would be a waste of effort and a potential of causing conflicting changes being made to a system.

Bug triagers need to manually go through the list of bug reports to detect if they are duplicate or not. This takes much effort. It has been reported in 2005 that for Mozilla "everyday almost 300 bugs appear that need triaging. This is far too much for only the Mozilla programmers to handle" [2]. Thus there is a need for an automated tool to help bug triagers make decision if a bug is a duplicate report or not. This work provides such a tool.

Why does helping bug triagers relevant for building or maintaining dependable systems? Aiding bug triagers job would make bugs processed more efficiently. Developers often have a limited amount of time before a product release. Many of bug reports take days even months to be closed. Thus, with improved efficiency in triaging bugs, more bugs could be fixed in a given amount of time. Thus, we believe improving bug triaging would contribute towards better maintenance of dependable systems.

To address the problem of duplicate bug reports, in the research community there have been two threads of work. One provides a solution to address the following problem:

> Given a new bug report, return other bug reports that are similar to it.

This thread of work include studies by Runeson et al. [18], Wang et al. [23], Sun et al. [21], [20], etc. We refer to this as *report retrieval* problem as the task is similar to the retrieval of similar document from a corpus. Another thread of work proposed by Jalbert and Weimer [10] addresses the following problem:

> Given a new bug report, classify it as either a duplicate bug report or not.

We refer to this as *report classification* problem as the task is to assign one out of the two labels, i.e. duplicate or not, to bug reports.

While the first thread of work (report retrieval) has received much attention, the latter (report classification) receives less. Indeed, to the best of our knowledge, there has been no study that extend the initial work by Jalbert and Weimer. We believe both threads of work complement one another in helping bug triagers do their job better. Report retrieval helps bug triagers perform manual identification of duplicate reports. Report classification could more fully *automate* the identification of duplicate reports. With report

1534-5351/12 $26.00 © 2012 IEEE
DOI 10.1109/CSMR.2012.48

retrieval and classification, triagers not only can see the top-k similar reports but also some flags to identify suspicious bug reports that are likely to be duplicates.

Jalbert and Weimer compute the similarity between a bug report with other reports. The highest similarity score (the similarity between a bug report to its nearest "neighbour") is used as a feature along with other *surface* features such as the component where the bug occurs, the day of the week, etc. A similarity measure based on the frequency of common terms appearing in the report is used in the study. They also cluster bug reports and use it as features with the intuition that clusters with one member tend to contain non-duplicate reports.

To improve the work by Jalbert and Weimer, we consider a number of extensions. First, rather than considering the frequency of common terms as a similarity measure we use an extension of BM25F [17], [24]. BM25F has been used widely in the information retrieval community and has been shown to outperform term frequency based similarity measure. The extension of BM25F has also been shown to be the most accurate measure for report retrieval [20]. Second, we consider an additional feature namely the differences in the "product" field of different bug reports. Bug reports specifying problems for different "product" are likely to be non-duplicate of one another. Third, rather than analyzing just top-1 most similar report, we build a composite feature that look into top-k most similar reports. We would like to compute a measure we call *relative similarity* which measures the relative similarity of a bug report to the top-1 with respect to the distances of other similar bug reports. Our intuition is that similarity distance varies depending on the topic of bug reports and we use distances between other similar reports to decide the right threshold when similar bug reports become duplicate bug reports.

We have experimented our approach on bug reports from Mozilla bug tracking system which were reported between February 2005 to October 2005, and show that our approach could outperform the previous study by Jalbert and Weimer. We increase the number of bug reports detected (or true positive rate) by 200% (from 8% to 24%) while keeping the number of false positives low (at 9%), i.e., a true negative rate of 91%. Computing the harmonic mean of true positive rate and true negative rate, we improve the accuracy of the previous approach by 160% (from 14.8% to 38.6%).

Our contributions are as follows:

1) We integrate the latest technique in bug report retrieval for improved bug report classification.
2) We introduce a new concept of relative similarity to differentiate duplicate bug reports from similar bug reports.
3) We have experimented with our solution on a dataset of bug reports from Mozilla bug tracking system and show that our approach could outperform the state-of-the-art approach proposed by Jalbert and Weimer.

Table I
BUG REPORT 299445 FROM MOZILLA BUG TRACKING SYSTEM

Summary	zlib buffer overflow
Description	I've discovered a deflate data stream that causes zlib to overwrite the bounds of an array, I believe this to be potentially exploitable under certain conditions to execute arbitrary code. Mark Adler, co-author of zlib, has provided the detailed explanation and proposed patch attached . . .
Product	core

The paper is organized as follows. In Section II, we introduce Bugzilla and the bug reporting process. In Section III, we present our proposed approach. In Section IV, we present our experiments. We present related work in Section V. We conclude and present future work in Section VI.

II. BACKGROUND

In this section, we introduce bug reporting process using Bugzilla and highlight the state-of-the-art study on report classification.

A. Bug Reporting Process

When a defect manifests itself or a feature request is raised, a user or tester can file a report to the corresponding software project. A bug report is a structured document consisting of several fields, $e.g.$, *summary*, *description*, *product*, etc. The summary field is a short description of the issue, while the description field is a more detailed description of the issue and how to reproduce it. Product field informs the product that is affected by the issue. We show an example bug report for Mozilla in Table I.

Once the new report arrives at the report repository, a bug triager needs to manually identify whether the report is a duplicate of an existing report in the repository or a new one. If it is a duplicate, the report is labeled DUPLICATE. Some bug reports are rejected (e.g., marked as WORKSFORME or INCOMPLETE, etc.). Otherwise, a new bug report is assigned to a developer for further investigation.

B. Report Classification: Jalbert & Weimer Approach

Jalbert and Weimer consider the same problem as ours. In their approach, they consider several features and build a linear model to classify whether a bug report is a duplicate or not. Several features that they use include:

1) Textual similarity. They consider textual similarity between a bug report and all existing bug reports (by comparing their titles and descriptions) and use the maximum of the similarity scores as two features (one for title and another for description).
2) Surface features. They consider various surface features of a bug report including: operating system, the day the bug report was made, the length of the title, whether the bug report has patch, and whether the severity is major.
3) Clustering. They form a graph where bug reports are nodes and the similarities between pairs of bug reports

Table II
A PAIR OF DUPLICATE BUG REPORTS

Id	Summary	Product
543839	JS_Assert failure "Thin_GetWait(tl->owner)" ...	core
543903	Assertion failure: Thin_GetWait(tl->owner) ...	core

are weights to edges connecting them. They cluster bug reports using a graph clustering algorithm by Mishra et al. [15]. They use the result of the clustering algorithm as a feature.

In their experiment they show that textual similarity is the most important feature while clustering is the least important feature. Their experiment on Mozilla bug reports has shown that their approach could identify duplicate bug reports with a true positive rate of 8% and true negative rate of 100%[1].

III. PROPOSED APPROACH

In this section, we describe our proposed approach. We first describe our set of features that could help to differentiate similar bug reports from duplicate bug reports. We then present the issue of imbalance data and how we handle that. Finally, we describe our overall approach.

A. Feature Engineering

We want to capture the pertinent aspects of a bug report to decide if it is likely a duplicate bug report or not. Intuitively, the more similar it is with another bug report, the more likely it is to be a duplicate bug report. Also, if two bug reports are referring to different parts of a large software system, unless the bug reports are wrongly reported, they are not likely to be duplicate reports. Furthermore, similarity is a relative concept. For some types of defects, high similarity might not mean that two reports are duplicates; for others, even two reports that are marginally similar are very likely to be duplicates. We make use of these three intuitions to formulate three features that we use to train a machine learning model to classify if a bug report is a duplicate or not. Table II shows a pair of duplicate bug reports from Mozilla bug tracking system – we observe that they are textually similar and refer to the same part of Mozilla, i.e., core.

1) Improved Similarity Metric: Jalbert and Weimer make use of term frequency of co-occurring words in two bug reports to detect how close two bug reports are. In the information retrieval community, term frequency and its extension, namely term frequency - inverse document frequency (TF-IDF), has been shown to be effective; however, a newer approach named BM25F has been proposed [17], [24]. Recently, Sun et al. has also shown that an extension of BM25F, namely REP has been shown to be the best measure for report retrieval problem [20].

REP considers a bug report as a structured document comprising of textual features, categorical features, and ordinal features. The textual features include:

1) The bag of words appearing in the title of a bug report.
2) The bag of digrams (i.e., two consecutive words) appearing in the title of a bug report.
3) The bag of words appearing in the description of a bug report.
4) The bag of digrams appearing in the description of a bug report.

The categorical features include information on the product and component that the bug is found on, along with the type of the report. The ordinal features include information on the version that the bug affects and the priority of the bug report.

In this work, rather that computing similarity score using term frequency for the title and description of the report *à la* Jalbert and Weimer, we compute a unified similarity score using REP. Given two bug reports B_1 and B_2, we denote the REP similarity score of B_1 and B_2 as $REP(B_1, B_2)$.

For a new bug report B_N and a bug report repository $Repo$, we compute the following measure:

$$Max\_Sim(B_N) = Max_{B_I \in Repo} REP(B_N, B_I)$$

2) Product Difference: To capture our second intuition, we single out product information from the other ordinal features used in computing similarity using REP and create a new feature. This feature is simply a boolean feature whose value is either 0 or 1, depending if two bug reports have different or the same product information. We compute the product difference feature from the new report B_N and the report most similar to it; We denote this as $Prod\_Diff(B_N)$.

3) Relative Similarity: To capture our third intuition, we need a metric to measure whether the similarity score between a new bug report B_N and its nearest bug report is significant or not. The intuition is that if many other bug reports are *equally* similar to the new bug report, but they themselves are not duplicate of one another, then the new bug report is likely not a duplicate too.

To compute such a metric, we extract top-k nearest bug reports (measured using REP) to the new bug report which are not duplicates of one another. For the 2^{nd} to the k^{th} nearest bug reports, we have $k-1$ similarity values. We then take the average of these $k-1$ values and compare it with the similarity of the new bug report with the top-1 nearest bug report (i.e., $Max\_Sim(B_N)$). If the average is very low as compared to $Max\_Sim(B_N)$, then the new bug report B_N is likely a duplicate as it is very similar to one bug report but not others. Mathematically, given the set of top-2 to top-k most similar bug reports TOP, this metric or feature is computed as:

$$Rel\_Sim(B_N) = \frac{Max\_Sim(B_N)}{Avg_{B_I \in TOP} REP(B_N, B_I)}$$

By default, we set the value of k to be 20.

[1]Please refer to Section IV-A for more details on how these measures are computed.

4) Overall Features: For each new bug report B_N, we thus characterize and represent it as a triple:

$$(Max\_Sim(B_N), Prod\_Diff(B_N), Rel\_Sim(B_N))$$

B. Addressing Imbalance Data

There are more bug reports that are non-duplicates than those that are duplicates. Thus we have an issue of imbalance dataset. Imbalance dataset causes an issue with a machine learner, as it would then be biased to pick the label of the majority. In our case, the machine learner would have the tendency to label every new bug reports as a non-duplicate report.

To address imbalanced data, two approaches are possible. One approach would be to reduce the training instances from the majority class (i.e., in our case, non-duplicate reports). Another approach would be to duplicate training instances from the minority class (i.e., in our case, duplicate reports). As the first approach would cause the loss of information (since we drop some training instances), we use the second approach and duplicate the duplicate reports so that we have a balanced dataset.

C. Overall Approach

Our overall approach is divided into training and deployment phase. In the training phase, we train a machine learning model based on a set of training data. In the deployment phase, we apply this model to label new bug reports as duplicate or not. The steps for the training phase are as follows:

1) We split our training data into two parts. One part is used to train the parameters of REP. To use REP to measure the similarity between two bug reports, some parameters need to be set; these parameters are weights to the different features of bug reports, i.e., textual, ordinal, and categorical features. The other part is used to train the machine learning model.
2) For the latter part of the training data, each bug report is converted to a triple using the feature engineering strategy presented in Section III-A.
3) We rebalance the training dataset using the approach presented in Section III-B.
4) We then train a machine learning model using Support Vector Machine (SVM) [8], [11].

IV. Experiments

In this section, we present our evaluation measures, our dataset, experiment results, and threats to validity.

A. Evaluation Measures

We follow the approach in Jalbert and Weimer's work [10] to identify true and false positives. Based on these, true positive rate (aka. sensitivity or recall rate) and true negative rate (aka. specificity) could be defined [13], [4]. We take the harmonic mean of true positive rate and true negative rate as the final evaluation criteria.

True positives are duplicate bug reports that are correctly identified by the proposed approach. Let's label this set as TP. Considering each group of reports that are duplicates of one another being put in the same *bucket*, the false positives are the number of buckets of duplicate bug reports for which there is not even one member of the bucket that is flagged as a non-duplicate. Let's label this set as FP.

Considering $Duplicate$ as the set of all duplicates, the true positive rate $TPRate$ is defined as:

$$TPRate = \frac{|TP|}{|Duplicate|}$$

Considering $NonDuplicate$ as the set of non-duplicate bug report buckets that get fixed, the true negative rate $TNRate$ is defined as:

$$TNRate = \frac{|NonDuplicate \setminus FP|}{|NonDuplicate|}$$

As developers are often overwhelmed with the number of bug reports, c.f., [2], both $TPRate$ and $TNRate$ are important. Both poor $TPRate$ and $TNRate$ can potentially cause some important reports to be missed – due to wasted effort and a limited resource, and incorrect identification of duplicate reports. We take the harmonic mean of true positive and true negative rate as the final evaluation criteria. It is defined by the following formula:

$$Harmonic = \frac{2 \times TNRate \times TPRate}{TNRate + TPRate}$$

B. Dataset

We follow the experimental setup of Jalbert and Weimer [10]. We collected bug reports from Mozilla bug tracking system which were reported between February 2005 to October 2005. Half of this dataset is used for training, while the other half for testing. Out of the half used for training, the first m reports that contain 200 duplicate bug reports are used to train REP. The remaining reports in the training set are used to train the machine learning model using SVM. Note that, we only perform dataset rebalancing as described in Section III-B on the training data used to train the machine learning model. For the testing data, we keep its original distribution, i.e., imbalanced.

C. Experiment Results

We compare our approach with that of Jalbert and Weimer and evaluate them in terms of true positive rate, false positive rate, and their harmonic mean. We compare our results with those reported in their paper [10]. We first present the overall results and then describe some additional experiments that we performed to investigate the effects of the parameter k and the relative importance of each of the three features.

Table III
OVERALL RESULTS: OURS VERSUS THE PRIOR STUDY

Approach	TPRate	TNRate	Harmonic
Ours	24.48%	91.40%	38.62%
Jalbert and Weimer's	8%	100%	14.8%

Table IV
EFFECT OF VARYING k

k	TPRate	TNRate	Harmonic
4	24.57%	91.95%	38.78%
8	24.48%	92.42%	38.71%
12	24.75%	91.91%	39.00%
16	24.66%	91.58%	38.86%
20	24.48%	91.40%	38.62%

1) Overall Results: The overall results of comparing our approach with that of Jalbert and Weimer's are shown in Table III. The true positive rate is increased from 8% to 24%. However, the true negative rate is reduced slightly from 100% to 91%. We notice a 200% increase in true positive rate for a 9% reduction in true negative rate. Our approach could increase the harmonic mean from 14.8% to 38.6% (a 160% improvement). This shows that our approach outperforms that of Jalbert and Weimer which is the state-of-the art approach in *report classification*.

2) Effect of Varying k: Our relative similarity feature (i.e., $Rel\_Sim(B_N)$) takes as input the parameter k which specifies the number of top-k similar bug reports to consider. In Section IV-C1, we use the default value of k (k=20). Here, we perform a sensitivity analysis on the value of k to see its effect on the performance of our proposed approach.

The result of varying k from 4 to 20 is shown in Table IV. We notice that varying k from 4 to 20 has not much impact on the true positive rate, true negative rate, and the harmonic mean of the two. We do not try k larger than 20, as the reports would then be significantly different.

3) Effectiveness of Different Features: We compute Fisher score to evaluate the effectiveness of different features in discriminating duplicate bug reports from non-duplicate ones. Fisher score is a popular measure used in statistics [6]. The Fisher score of a feature is defined as:

$$Fisher = \frac{\Sigma_{i=1}^{c} n_i(\mu_i - \mu)^2}{\Sigma_{i=1}^{c} n_i \sigma_i^2}$$

where n_i is the number of instances or data points in class i (i.e., the number of bug reports in a dataset), μ_i is the average value of the feature in class i (i.e., duplicate or non-duplicate), σ_i is the standard deviation of the feature values in class i, and μ is the average value of the feature in the whole dataset.

Table V show the Fisher scores for the three features. From the scores we could note that relative similarity ($Rel\_Sim(B_N)$) has the highest Fisher score. The second is maximum similarity ($Rel\_Sim(B_N)$). Both of them are much more discriminative than product difference ($Prod\_Diff(B_N)$).

Table V
FISHER SCORE FOR THE 3 FEATURES

Feature	Fisher Score
$Rel\_Sim(B_N)$	0.157
$Max\_Sim(B_N)$	0.138
$Prod_Diff(B_N)$	0.011

D. Threats to Validity

As with many other studies, there are a number of possible threats to validity including threat to internal validity, threat to external validity, and threat to construct validity.

Threat to internal validity includes experimenter bias. In this study, we choose a subset of the dataset chosen by Jalbert and Weimer. We believe there is little bias as the dataset is simply a set of tens of thousands of bug reports from a well known software system.

Threat to external validity corresponds to the generalizability of our experimental results. We have experimented with bug reports from Mozilla bug tracking system which were reported between February 2005 to October 2005. In the future, we would consider more software systems and a larger set of bug reports.

Threat to construct validity refers to the suitability of our metric. We use the same notions of true positive and true negative defined by Jalbert and Weimer. From these notions of true and false positive, we compute standard measures in information retrieval and statistics namely true positive rate (aka. specificity or recall rate) and true negative rate (aka. sensitivity). We combine these two measures into a unified measures by taking their harmonic mean. We believe these measures (true positive rate, true negative rate, and their harmonic mean) are reasonable; the more true positives (i.e., duplicate found) and the more true negatives (i.e., fixed non-duplicate report identified) the better is an approach.

V. RELATED WORK

The pioneer study on bug reports classification is by Jalbert and Weimer [10], which is also closest to ours. They report that 8% of the duplicate reports could be filtered by their technique in the experiments ($TPRate$ of 8% and $TNRate$ of 100%). As shown in Section IV, our approach can filter duplicate reports with $TPRate$ of 24.48% and $TNRate$ of 91.40% – 160% relative improvement in terms of $Harmonic$. Generally, the high true positive rate stems from a classification model trained with more effective features. In particular, first, our approach is based on a better similarity measure REP specially designed for measuring similarity of bug reports [20]. Whereas, Jalbert and Weimer use *cosine*, a similarity measure based on TF-IDF for general documents. Second, we consider the difference between *product* fields of reports, as reports on different products tend to be non-duplicates of one and each. Last but not least, we propose the concept of *relative similarity* to further improve the classification accuracy.

Another line of research is retrieval of duplicate reports. When a new report comes, it returns a list of reports

existing in the repository which are potentially similar to the new report. The intrinsic problem is how to measure the similarity between two reports. Runeson *et al.* take natural language text of bug reports and use $cosine$, $dice$ and $jaccard$ to measure the similarity of reports [18]. Wang *et al.* use not only natural language text bug also execution information to improve the retrieve performance [23]. Sun *et al.* propose a machine learning approach and extend BM25F to accurately retrieve duplicate reports [21], [20]. Our work is orthogonal to the above as we aim to output a binary answer DUPLICATE or NEW for a new report.

Besides the effort on bug report classification and retrieval, there are also other studies on report categorization. Anvik et al. [1], Cubranic and Murphy [5], Pordguski et al [16], Francis et al. [7], and Tamrawi et al. [22] all investigate the problem of bug report categorization for better triaging. In [1], [5], [22], this categorization is used to assign bug reports to the right developers. Menzies and Marcus [14] propose a prediction model to automatically infer the severity of bug reports. Ko and Myers investigate the differences between defect reports and feature requests based on the linguistic characteristics of summaries and descriptions in bug reports [12].

Researchers have also done several empirical studies on bug repositories. Sandusky et al. investigate the nature, extent and impact of bug report networks in one large F/OSS development community [19]. Anvik et al. empirically study the characteristics of bug repositories and show interesting findings on the number of reports that a person submitted and the proportion of different resolutions [2]. Hooimeijer and Weimer develop a descriptive model based on a statistical analysis of surface features of over 27,000 bug reports in OSS projects, to predict bug report quality [9]. Bettenburg et al. survey developers of Eclipse, Mozilla, and Apache to study what makes a good bug report. A good bug report provides enough information to developers for debugging [3].

VI. Conclusion and Future Work

Bug reporting is an uncoordinated distributed process. Thus, often there are many duplicate reports being submitted that report the same defect. To address this issue, there is a need for an automated duplicate report detection approach. There are two families of research work in this direction: report retrieval and report classification. In this work, we extend the latest study on report classification by Jalbert and Weimer. We extend their approach by utilizing REP which was recently proposed for report retrieval problem to measure the similarity of two bug reports. We also utilize information on the difference between product fields in two bug reports to help in identifying if two bug reports are duplicate or not. Furthermore, we define a new notion of relative similarity that help to decide if the similarity between two bug reports is significant enough. We have performed experiments on Mozilla's bug reports which were reported between February 2005 to October 2005. Our preliminary study has shown that our approach is effective to increase the true positive rate by 200% (from 8% to 24%) while only suffering a loss in true negative rate by 9% (from 100% to 91%). The overall harmonic mean of true positive rate and true negative rate is increased by 160% (from 14.8% to 38.62%).

In the future, we plan to extend this study by investigating more bug reports from different software systems. We also would like to continue improving the accuracy of duplicate bug report classification further. We also plan to release a tool that would help developers to flag duplicate bug reports and integrate it to bug management systems such as Bugzilla.

References

[1] J. Anvik, L. Hiew, and G. Murphy, "Who should fix this bug?" in *ICSE*, 2006.

[2] J. Anvik, L. Hiew, and G. C. Murphy, "Coping with an open bug repository," in *ETX*, 2005.

[3] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *FSE*, 2008.

[4] M. Bland, *An Introduction to Medical Statistics*. Oxford, 2000.

[5] D. Cubranic and G. C. Murphy, "Automatic Bug Triage Using Text Categorization," in *SEKE*, 2004.

[6] R. Duda, P. Hart, and D. Stork, *Pattern Classification*. Wiley Interscience, 2000.

[7] P. Francis, D. Leon, and M. Minch, "Tree-based methods for classifying software failures," in *ISSRE*, 2004.

[8] J. Han and M. Kamber, *Data Mining Concepts and Techniques*, 2nd ed. Morgan Kaufmann, 2006.

[9] P. Hooimeijer and W. Weimer, "Modeling bug report quality," in *ASE*, 2007.

[10] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," in *DSN*, 2008.

[11] T. Joachims, http://svmlight.joachims.org.

[12] A. Ko and B. Myers, "A linguistic analysis of how people describe software problems," in *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2006.

[13] C. Manning, P. Raghavan, and H. Shutze, *Introduction to Information Retrieval*. Cambridge, 2008.

[14] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in *ICSM*, 2008.

[15] N. Mishra, R. Schreiber, I. Stanton, and R. Tarjan, "Clustering social networks," in *Workshop on Algorithms and Models for the Web-Graph (WAW)*, 2007.

[16] A. Podgurski, D. Leon, P. Francis, W. Masri, M. Minch, J. Sun, and B. Wang, "Automated support for classifying software failure reports," in *ICSE*, 2003.

[17] S. Robertson, H. Zaragoza, and M. Taylor, "Simple BM25 extension to multiple weighted fields," in *ACM International Conference on Information and Knowledge Management (CIKM)*, 2004.

[18] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in *ICSE*, 2007.

[19] R. J. Sandusky, L. Gasser, R. J. S, U. L. Gasser, and G. Ripoche, "Bug report networks: Varieties, strategies, and impacts in a f/oss development community," in *MSR*, 2004.

[20] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," in *ASE*, 2011.

[21] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in *ICSE*, 2010.

[22] A. Tamrawi, T. T. Nguyen, J. Al-Kofahi, and T. N. Nguyen, "Fuzzy set-based automatic bug triaging (NIER track)," in *ICSE*, 2011.

[23] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *ICSE*, 2008.

[24] H. Zaragoza, N. Craswell, M. Taylor, S. Saria, and S. Robertson, "Microsoft cambridge at trec 13: Web and hard tracks," in *Text Retrieval Conference (TREC)*, 2004.