Review of procession of research of duplicate bug report detection

Bhashwanth Kadapagunta
North Carolina State University
bkadapa@ncsu.edu

Venkatesh Sambandamoorthy
North Carolina State University
vsamban@ncsu.edu

ABSTRACT
Bug tracking systems are important tools that guide the maintenance activities of software developers. The utility of these systems is hampered by an excessive number of duplicate bug reports–in some projects as many as a quarter of all reports are duplicates. Developers must manually identify duplicate bug reports, but this identification process is time-consuming and exacerbates the already high cost of software maintenance. A system that automatically classifies duplicate bug reports as they arrive could save developer time. In this paper, we review the procession in the development of such systems during the period of 2008-2015 by examining the related literature. We examine how the various methods proposed by different papers contribute towards achieving the goal of automatic duplicate bug report detection.

Keywords
Bug tracking system, Duplicate Bugs, Triager, Software maintenance, Developer Recommendation

1. INTRODUCTION

Due to complexities of software systems, software bugs are prevalent. To manage and keep track of bugs and their associated fixes, bug tracking system like Bugzilla has been proposed and is widely adopted. Defect reporting is an integral part of a software development, testing and maintenance process. Typically, bugs are reported to an issue tracking system which is analyzed by a Triager (who has the knowledge of the system, project and developers) for performing activities like: quality check to ensure if the report contains all the useful and required information, duplicate detection, routing it to the appropriate expert for correction and editing various project-specific metadata and properties associated with the report (such as current status, assigned developer, severity level and expected time to closure). It has been observed that often a bug report submitted by a tester is a duplicate (two bug reports are said to be duplicates if they describe the same issue or problem and thereby have the same solution to fix the issue) of an existing bug report. Studies show that the percentage of duplicate bug reports can be up-to 25-30% [8][9][10].

To address this issue there have been a number of studies that try to partially automate the triaging process. There are two general approaches: one is by filtering duplicate reports preventing them from reaching the triagers [2], the other is by providing a list of top-related bug reports for each new bug report under investigation [3]–[6]. Developer time and effort are consumed by the triage work required to evaluate bug reports, and the time spent fixing bugs has been reported as a useful software quality metric. Modern software engineering for large projects includes bug report triage and duplicate identification as a major component. Duplicate bug reports could potentially provide different perspectives to the same defect potentially enabling developers to better fix the defect in a faster amount of time. Still there is a need to detect bug reports that are duplicate of one another.

Duplicate Bug Reports

A typical bug report contains many fields, such as reporter, fixer, creation time, modification time, bug version, platform, CC

list, comment list, etc. In this work, we collected 5 pieces of information from the bug report fields including: bug summary, bug description, product affected by the bug, component affected by the bug, and developers participated in the bug resolution process (i.e., bug resolvers). The details of these pieces of information is presented Table I.

Info.	**Details**	**Example**
Summary	Brief description of a bug.	frequent ”invalid thread access”
Description	Detailed description of a bug.	I’m not sure 100% where the problem
Product	Product affected by the bug.	Platform
Component	Component affected by the bug.	SWT
Developers	Bug resolvers	Steffen Pingel, Mik Kersten

Table I: Contents of a bug report

Duplicate report retrieval system

In a duplicate report retrieval system, the fields’ summary and description of both existing and new bug reports are preprocessed by standard information retrieval techniques, tokenization, stemming and stop work removal. Figure 1 depicts the overall flow. The bug repository is organized as a list of buckets – a hash map-like data structure. The key of each bucket is a master report, and its value is a list of duplicate reports on the same bug. Each bucket has a distinct bug as its master report is not contained in other buckets. When a fresh bug report is submitted, the system computes the similarity between and each bucket, and returns master reports, whose buckets have the top- similarities. The similarity of a report and a bucket is the maximum similarity between the report and each report in the bucket computed by the component Similarity Measure in Figure 1.

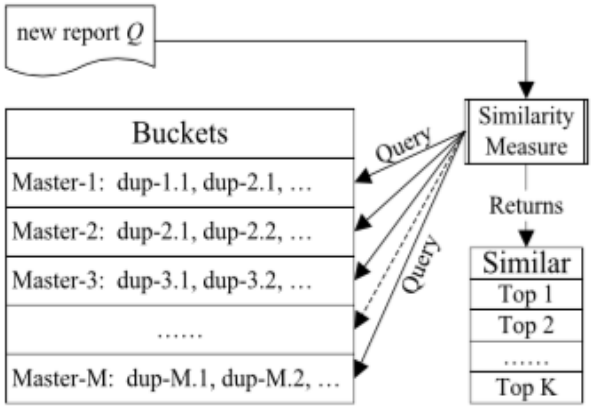


Figure 1: Duplicate bug retrieval system

2. MOTIVATION

The “Most Frequently Reported Bugs” page of the Mozilla Project’s Bugzilla bug tracking system tracks the number of bug reports with known duplicates and displays the most commonly reported bugs.

In the Firefox bug repository, both Bug-260331 and Bug-239223 are about the incomplete closing of the browser. Bug-260331 had been identified as a duplicate of Bug-239223. Specifically, their summaries are as follows.

Bug-260331: After closing Firefox, the process is still running. Cannot reopen Firefox after that, unless the previous process is killed manually

Bug-239223: (Ghostproc) – [Meta] firefox.exe doesn't always exit after closing all windows; session-specific data retained

Both summaries share words like “firefox” and “after closing”, but as these words are very common in the Firefox bug repository, it is difficult to use these words to confirm a duplicate relationship between the two bug reports.

Bug report #340535 is indicative of the problems involved; we will consider it and three of its duplicates.The body of bug report #340535, submitted on June 6, 2006, includes the text, “when I click OK the updater starts again and tries to do the same thing again and again. It never stops. So I have to kill the task.” It was reported with severity “normal” on Windows XP and included a logfile.

Bug report #344134 was submitted on July 10, 2006 and includes the description, “I got a software update of Minefield, but it failed and I got in an endless loop.”. Another report, #372699, was filed on March 5, 2007. It included the title, “autoupdate runs...and keeps doing this in an infinite loop until you kill thunderbird.”

When these three example reports are presented in succession their similarities are evident, but in reality they were separated by up to nine months and over thirty thousand intervening defect reports. All in all, 42 defect reports were submitted describing this same bug. In commercial development processes with a team of bug triagers and software maintainers, it is not reasonable to expect any single developer to have thirty thousand defects from the past nine months memorized for the purposes of rapid triage. Developers must thus be wary, and the cost of checking for duplicates is paid not merely for actual duplicates, but also for every non-duplicate submission; developers must treat every report as a potential duplicate.

3. RELATED WORK

One of the pioneering studies on duplicate bug report detection is by Runeson et al. [15]. Their approach first cleaned the textual bug reports via natural language processing techniques – tokenization, stemming and stopword removal. The remaining words were then modeled as a vector space, where each axis corresponded to a unique word. Each report was represented by a vector in the space. The value of each element in the vector was computed by the formula on the tf value of the corresponding word. After these vectors were formed, they used three measures – cosine, dice and jaccard – to calculate the distance between two vectors as the similarity of the two corresponding reports. Given a bug report under investigation, their system would return top-k similar bug reports based on the similarities between the new report and the existing reports in the repository. A case study was performed on defect reports at Sony Ericsson Mobile Communications, which showed that the tool was able to identify 40% of duplicate bug reports. It also showed that cosine outperformed the other two similarity measures.

Mining software repositories is an emerging field that has received significant research interest in recent times. Hiew et al. presents an approach based on grouping similar reports in the repository and deriving a centroid of the clustered reports [16]. An incoming report (represented as a document vector) is then compared to the centroids of bug report groups to compute a similarity score for detecting duplicates based on a predefined threshold value. One of the unique features of the approach presented by Hiew et al. is pre-processing of bug reports to create a model (model updated as new bug reports arrived) consisting bug-report groups and their respective centroid which is then used for similarity computations. Hiew et al. perform experiments on bug report data from open-source projects such as Eclipse and Firefox project [16].Kim and Whitehead claim that

the time it takes to fix a bug is a useful software quality measure [20]. They measure the time taken to fix bugs in two software projects.

Sandusky et al. study the statistics of duplicate bug reports [12]. Hooimeijer and Weimer develop a model to predict bug report quality [13]. Bettenburg et al. survey developers of Eclipse, Mozilla, and Apache to study what developers care most in a bug report [14]. Bettenburg et al. later show that duplicate bug reports might be useful [11]. Anvik et al. [17], Cubranic and Murphy [18] and Lucca et al. [19] all proposed semi-automatic techniques to categorize bug reports.

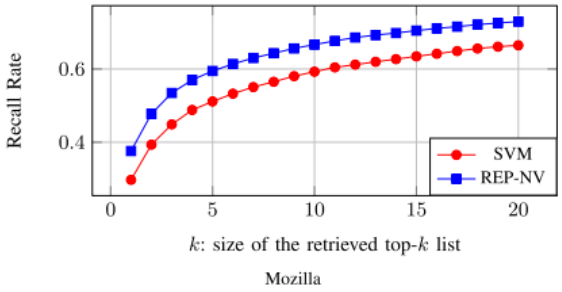
4. IMPLEMENTATION

We selected a highly cited paper that was published in 2011 by Sun, Chengnian, et al.[1] which dealt with the issue of duplicate bug report detection. We reviewed this paper to analyse the state of the art on this issue at that point. We then chose other papers from among the literature that were referenced in the first paper. A total of 4 such papers were chosen going backwards in time from 2011 to 2008. Analysing these papers gave us a clear view of the background work that had been done in this field and the progression that was happening each year. We then chose papers that were published during the period of 2011 to 2015 that referenced the first paper. A total of 3 such papers were reviewed. Analysis of these papers helped us track the further progress of research happening in this field and get a sense of the future direction of research.

5. RESULTS

In this section, we review the procession in research in the field of duplicate bug report detection during the period of 2008 to 2015 by examining the related literature. We also examine how the various methods proposed by different papers contribute towards achieving the goal of automatic duplicate bug report detection. A total of 8 papers were reviewed ranging from 2008 to 2014 and the progress at each step was analyzed.

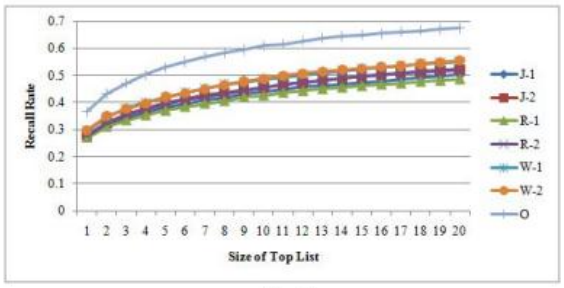
The review process started with the paper titled “Automated Duplicate Detection for Bug Tracking Systems” by N.Jalbert et al[5]. The paper proposes a system that automatically classifies duplicate bug reports as they arrive to save developer time. This system uses surface features, textual semantics, and graph clustering to predict duplicate status. Evaluation was done using a dataset of 29,000 bug reports from the Mozilla project and was shown that inverse document frequency is not useful in this task.



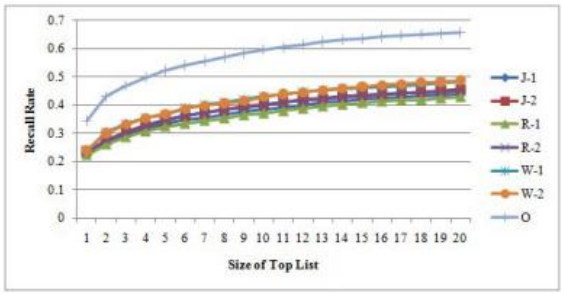
The developed model was used as a filter in a real-time bug reporting environment and was able to reduce development cost by filtering out 8% of duplicate bug reports. It still allows at least one report for each real defect to reach developers, and spends only 20 seconds per incoming bug report to make a classification.

X.Wang et al. continued the research on this topic in their work “An Approach to Detecting Duplicate Bug Reports using Natural Language and Execution Information”[4]. They present a novel approach to assist triagers in detecting duplicate bug reports. Unlike existing approaches which exploit only natural language information to detect duplicate bug reports, this approach further considers execution information. Furthermore, the approach employs two heuristics to combine the two kinds of

information. Evaluated of the approach was done using bug reports from the Eclipse and Firefox repositories.



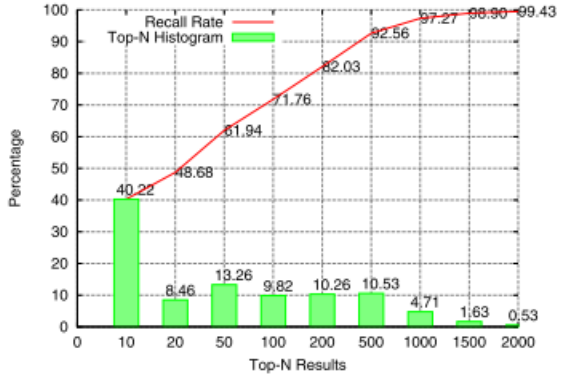
FireFox



Eclipse

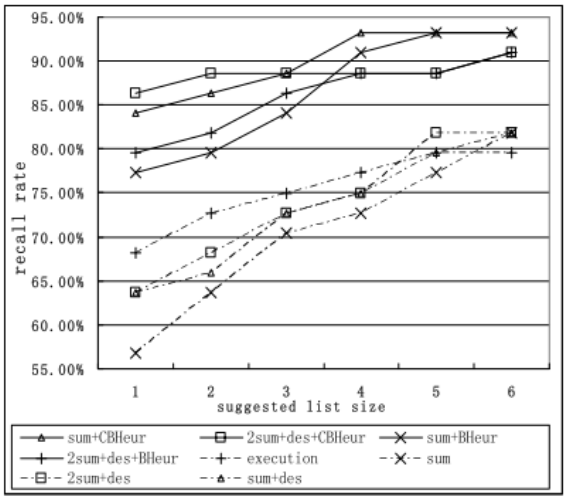
The experimental results show that, compared with the best performance of approaches using only natural language information, the calibrated approach (with the classification-based heuristic and using only the summary) leads to an increase of 11-20 percentage points and an increase of 18-26 percentage points in recall rates on the two experimental bug-report sets respectively.

A.Sureka et al. present an approach to compute text similarity between two bug reports to assist a triager in the task of duplicate bug report detection in their paper “Detecting Duplicate Bug Report Using Character N-Gram-Based Features”[3]. The central idea behind the proposed approach is the application of character n-grams as low-level features to represent the title and detailed description of a bug report. The advantages of the approach are language independence as it does not require language specific pre-processing and ability to capture sub-word features which is useful in situations requiring comparison of noisy text. The approach is evaluated on a bug database consisting of more than 200 thousand bug reports from open- source Eclipse project.



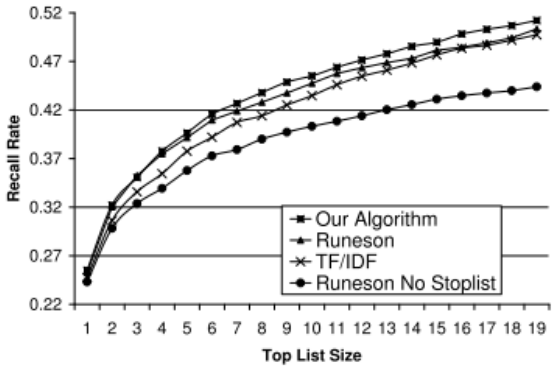
The recall rate for the Top 50 results is 33.92% for 1100 randomly selected test cases and 61.94% for 2270 randomly selected test cases with a title to title similarity (between the master and the duplicate) of more than a pre-defined threshold of 50.

The paper by C.Sun et al. titled “A Discriminative Model Approach for Accurate Duplicate Bug Report Retrieval”[2] presents a new approach to detecting duplicate bug reports by building a discriminative model that answers the question “Are two bug reports duplicates of each other?”. The model would report a score on the probability of A and B being duplicates. This score is then used to retrieve similar bug reports from a bug report repository for user inspection. The utility of the approach was investigated on 3 sizable bug repositories from 3 large open-source applications including OpenOffice, Firefox, and Eclipse.



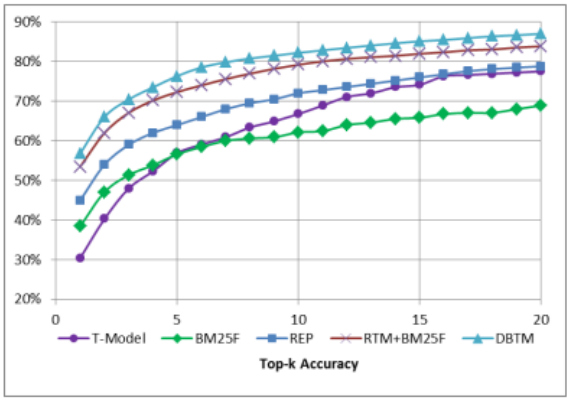
The experiment showed that the approach outperforms existing state-of-the-art techniques by a relative improvement of 17–31%, 22–26%, and 35–43% on OpenOffice, Firefox, and Eclipse dataset respectively.

In the paper titled “Towards More Accurate Retrieval of Duplicate Bug Reports”, C.Sun et al.[1] improve the accuracy of duplicate bug retrieval in two ways. First, BM25F is an effective textual similarity measure which is originally designed for short unstructured queries, and it is extended to BM25F ext especially for lengthy structured report queries by considering weight of terms in queries. Second, they propose a new retrieval function REP fully utilizing not only text but also other information available in reports such as product, component, priority etc. A two-round gradient descent contrasting similar pairs of reports against dissimilar ones, is adopted to optimize REP based on a training set. The utility of the technique was investigated on 4 sizable bug datasets extracted from 3 large open-source projects, i.e., OpenOffice, Firefox and Eclipse; and found that both BM25F ext and REP are indeed able to improve the retrieval performance.



Particularly, the experiments on the four datasets show that BM25F ext improves recall rate@k by 3-13% and MAP by 4-11% over BM25F. For retrieval performance of REP, compared to the previous work of the authors based on SVM, it increases recall rate@k by 10–27%, and MAP by 17–23%; compared to the work by Sureka and Jalote [3], REP performs with recall rate@k of 37–71% ($1 \le k \le 20$), and MAP of 46%, which are much higher than the results reported in their paper.

A.Nguyen et al. introduce DBTM, a duplicate bug report detection approach that considers not only IR-based features but also topic-based features in their paper “Duplicate Bug Report Detection with a Combination of Information Retrieval and Topic Modeling”[6]. DBTM models each bug report as a textual document describing one or more technical issues, and models duplicate bug reports as the ones on the same technical issue. Trained with historical data including identified duplicate reports, DBTM can learn sets of different terms describing the same technical issues and detect other not-yet-identified duplicate ones.



Accuracy Comparison in Eclipse

Empirical evaluation on real-world systems showed that DBTM can improve the state-of-the-art approaches by up to 20% in accuracy.

The research focus so far has been in the direction of improving the accuracy of duplicate bug detection. Now, X.Xia et al.[7] focus on developer recommendation for a particular bug in the paper "Accurate Developer Recommendation for Bug Resolution". They propose a new method DevRec to automatically recommend developers for bug resolution. They consider two kinds of analysis: bug report based analysis (BR-Based analysis) and developer based analysis (D-Based analysis). DevRec takes the advantage of both BR-Based and D-Based analysis, and compose them together. The experiment results show that, compared with other state-of-the-art approaches, DevRec achieves the best performance.

RECALL@5 AND RECALL@10 OF *DevRec* AND BR-BASED COMPONENT.

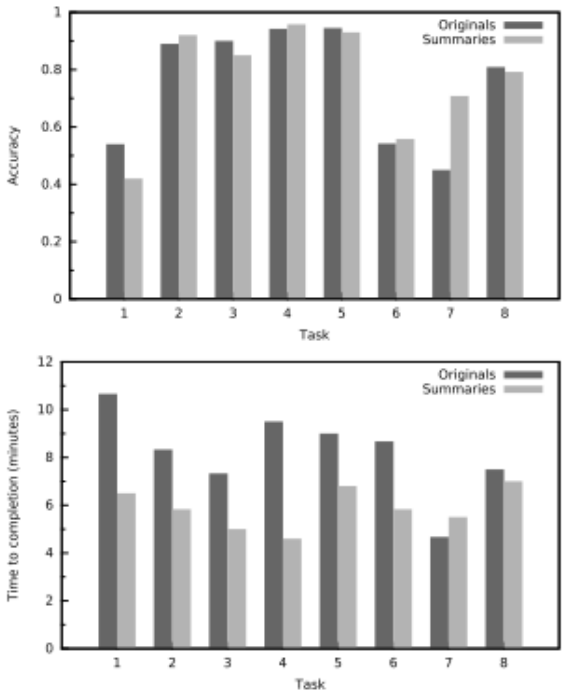
	Recall@5			Recall@10		
Projects	***DevRec***	**BR.**	**Improve.**	***DevRec***	**BR.**	**Improve.**
GCC	0.5633	0.4820	**16.87%**	0.7072	0.6490	**8.97%**
OpenOffice	0.4826	0.4670	**3.34%**	0.6063	0.5728	**5.85%**
Mozilla	0.5592	0.5487	**1.91%**	0.6755	0.6567	**2.86%**
Netbeans	0.7130	0.6974	**2.24%**	0.8082	0.7873	**2.65%**
Eclipse	0.7989	0.7873	**1.47%**	0.8924	0.8595	**3.93%**
Average.	0.6234	0.5965	**5.17%**	0.7379	0.7051	**4.83%**

RECALL@5 AND RECALL@10 OF *DevRec* AND D-BASED COMPONENT.

	Recall@5			Recall@10		
Projects	***DevRec***	**D.**	**Improve.**	***DevRec***	**D.**	**Improve.**
GCC	0.5633	0.5524	**1.97%**	0.7072	0.6952	**1.73%**
OpenOffice	0.4826	0.4783	**0.90%**	0.6063	0.6059	**0.07%**
Mozilla	0.5592	0.5053	**10.67%**	0.6755	0.6313	**7.00%**
Netbeans	0.7130	0.6383	**11.70%**	0.8082	0.7794	**3.70%**
Eclipse	0.7989	0.7602	**5.09%**	0.8924	0.8708	**2.48%**
Average.	0.6234	0.5869	**6.07%**	0.7379	0.7165	**2.99%**

DevRec improves the average recall@5 and recall@10 scores of Bugzie by 57.55% and 39.39%, respectively. DevRec also outperforms DREX by improving the average recall@5 and recall@10 scores by 165.38% and 89.36%, respectively.

The paper titled "Automatic Summarization of Bug Reports" by S.Rastkar[8] investigated the automatic generation of one kind of software artifact, bug reports, to provide developers with the benefits others experience daily in other domains. They found that existing conversation-based extractive summary generators can produce summaries for reports that are better than a random classifier. They also found that an extractive summary generator trained on bug reports produces the best results. They showed that generated bug report summaries could help developers perform duplicate detection tasks in less time with no indication of accuracy degradation, confirming that bug report summaries help software developers in performing software tasks.



6. AREAS OF IMPROVEMENT

In their research, Rastkar et al. apply support vector machine methods to summary bug reports [8]. To be admitted, support vector machine method is an effective approach in many areas, such as geophysics analysis and face recognition. On the other hand, it is not a judicious and advisability way in bug report analysis for the following reasons. It is necessary to compute a great number of 'vectors' (word weight) in bug reports, by this, it is easily to depend on a tendency using a very high level vectors. Though it maybe increases the accuracy in training set, there is a marked plunge rate in test set. On the contrary to that, naïve Bayes classifier can avoid the problem of support vector machine by statistical method.

The developer affinity score used by X.Xia et al. in their work[7] doesn't take into account the qualitative parameters of previously resolved bugs. In the paper "Duplicate Bug Report Detection with a Combination of Information Retrieval and Topic Modeling" by A.Nyugen et al.[6], the system execution conditions under which the bugs occurred were not taken into consideration.

7. CONCLUSION AND FUTURE WORK

Review of the procession of research in the field of duplicate bug report detection during the period of 2008 to 2014 showed that the focus was primarily on improving the accuracy of bug report classification. Various techniques based on textual semantics, graph clustering, natural language information, character n-grams, execution Information, BM25Fext have been developed and improved upon for this purpose through the research during this period. Towards the end of the review duration, the focus of some researchers moved from just improving the accuracy of classification by employing different techniques to helping improve the overall efficiency of the project by proposing techniques like automatic developer recommendation systems.

The research work reviewed here proposed techniques based on natural language information and execution information. They do not capture the context that the text is being used in. Techniques like unstructured text analysis could be used in the future to improve the accuracy of bug report classification. This analysis could help us detect bug reports which are discussing the same issue using different terminologies. Research work in the future can also focus on developing integrated classification approaches which use the bug report information as well as developer expertise to come up with better classification systems and also to improve the overall efficiency of the testing process in software engineering.

8. REFERENCES

1. Sun, Chengnian, et al. "Towards more accurate retrieval of duplicate bug reports." *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 2011.
2. Sun, Chengnian, et al. "A discriminative model approach for accurate duplicate bug report retrieval." *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. ACM, 2010.
3. Sureka, Ashish, and Pankaj Jalote. "Detecting duplicate bug report using character n-gram-based features." *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*. IEEE, 2010.
4. 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 proceedings of the International Conference on Software Engineering, 2008
5. N. Jalbert and W. Weimer, "Automated Duplicate Detection for Bug Tracking Systems," in proceedings of the International Conference on Dependable Systems and Networks, 2008.

6. A. Nyugen, D. Lo and C. Sun, "Duplicate Bug Report Detection with a Combination of Information Retrieval and Topic Modeling" in ASE 2012.
7. X. Xia et al "Accurate Developer Recommendation for Bug Resolution" in WCRE 2013.
8. S. Rastkar, G. Murphy "Automatic Summarization of Bug Reports" in IEEE TRANSACTIONS ON SOFTWARE ENGINEERING 2014.
9. John Anvik, Lyndon Hiew, and Gail C. Murphy. Who should fix this bug? In ICSE '06: Proceedings of the 28th international conference on Software engineering, pages 361– 370, New York, NY, USA, 2006. ACM.
10. Lyndon Hiew. Assisted detection of duplicate bug reports. In MS Computer Science Thesis, Department of Computer Science, British Columbia, Canada, 2003. The University of British Columbia.
11. N.Bettenburg,R.Premraj,T.Zimmermann ,andS.Kim,"Duplicatebug reports considered harmful ... really?" in ICSM08: Proceedings of IEEE International Conference on Software Maintenance, 2008, pp. 337–345.
12. 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 International Workshop on Mining Software Reposito- ries, 2004, pp. 80–84.
13. P. Hooimeijer and W. Weimer, "Modeling bug report quality," in ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, 2007, pp. 34–43.
14. N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zim- mermann, "What makes a good bug report?" in SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, 2008, pp. 308–318.
15. P. Runeson, M. Alexandersson, and O. Nyholm. Detection of Duplicate Defect Reports Using Natural Language Processing. In proceedings of the International Conference on Software Engineering, 2007.
16. Lyndon Hiew. Assisted detection of duplicate bug reports. In MS Computer Science Thesis, Department of Computer Science, British Columbia, Canada, 2003. The University of British Columbia.
17. Anvik, J., Hiew, L., and Murphy, G. Who Should Fix This Bug? In Proc. ICSE., 2006, 371-380.
18. Cubranic, D. and Murphy, G. Automatic Bug Triage Using Text Classification. In Proc. SEKE, 2004, 92-97.
19. Lucca, D., Penta, D., Granada, S., An Approach to Classify Software Maintenance Requests. In Proc. ICSM, 2002, 93-102.
20. S. Kim and J. E. James Whitehead. How long did it take to fix bugs? In International workshop on Mining Software Repositories, pages 173–174, 2006.