Developments in Duplicate Bug Reporting Process in a Software

Ahmad Saad Khan
North Carolina State University
Raleigh NC
akhan7@ncsu.edu

Neha Kale
North Carolina State University
Raleigh NC
nkale@ncsu.edu

Sekharan Natarajan
North Carolina State University
Raleigh NC
smnatara@ncsu.edu

ABSTRACT
Testers or users submit bug reports to identify various issues with systems. Bug repositories are the central storage for the bug reports. Sometimes two or more bug reports correspond to the same defect and thus gives duplicate bug reports. To address the problem with duplicate bug reports, a person called a triager needs to manually label these bug reports as duplicates, and link them to their master reports for subsequent maintenance work.
However, in practice there are considerable duplicate bug reports sent daily and requesting triagers to manually label these bugs could be highly time consuming.
We do have modern bug tracking systems 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. 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 2010-2016 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
Defect Localization; Abstract Syntax Tree (AST); Within-Project Defect Prediction (WPDP); Bug repositories

1. INTRODUCTION
Modern software systems are highly complex and hence software bugs are widely 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.

Typically, a bug report contains textual description of the issue and the method of reproducing the bug. It is a structured record consisting of many fields. Commonly, they include summary, description, project, submitter, priority and so forth. The figure below shows a typical bug report. Each field carries a different type of information. For example, summary is a concise description of the defect problem while description is the detailed outline of what went wrong and how it happened. Both of them are in natural language format. Other fields such as project, priority try to characterize the defect from other perspectives. A status field in a bug is a representation of the different stages in a lifecycle of a bug.

Studies show that the percentage of duplicate bug reports can be up-to 25-30% [13][10][6]. 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 [14], the other is by providing a list of top-related bug reports for each new bug report under investigation [15][1]. 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.

2. MOTIVATION

The process of bug reporting is still not mature and is uncoordinated and ad-hoc. It is very common that the same bugs could be reported more than once by different users. Hence, there is often a need for manual inspection to detect whether the bug has been reported before. If the incoming bug report is not reported before then the bug should be assigned to a developer. But if other users have reported the bug before then the bug should be classified as a ”duplicate” and should be attached to the original first - reported ”master” bug report. There are two challenges related to bug data that may affect the effective use of bug repositories in software development tasks and they are ”large scale” and ”low quality”. On one hand, due to the daily-reported bugs, a large number of new bugs are stored in bug repositories. Taking an open source project, Eclipse, as an example, an average of 30 new bugs were reported to bug repositories per day in 2007 [7]; from 2001 to 2010, 333,371 bugs were reported to Eclipse by over 34,917 developers and users [9][16].

The Mozilla programmers reported in 2005 that ”every day, almost 300 bugs appear that need triaging. This is far too much for only the Mozilla programmers to handle” [8]. This is seeming to be the most time consuming step of handling a bug and alleviating the burden of the triagers has gained a lot of importance over the past decade, in the field of automated software engineering. A lot of effort has been put in deploying an automated triaging system in the industry

Thus recognizing duplicate reports is an important problem that, if solved, would enable developers to fix bugs faster, and prevent them from wasting time by addressing the same bug multiple times. Classifying bug reports as duplicates by applying some of the Data Mining techniques of ”textual similarity” and ”information retrieval” has been a very popular approach.

3. DATASETS

Most of the studies have used the bug repositories of open source projects - OpenOffice, Mozilla and Eclipse as their data for conducting their study. These repositories are very popular because these projects are all open source and have a large repository of bug reports and are freely accessible. These projects are also very different from one another in terms of purposes, users and implementation languages, and thus help in generalizing the conclusions of the experiments.

In [11], Amoui et al. applied the Duplicate Defect Detection Framework which they build to the Blackberry bug repository in addition to the OpenOffice, Mozilla and Eclipse repositories.

4. RELATED WORK

There has been lot of relate work on duplicate bug detection in the past. One of the pioneering studies on duplicate bug report detection is by Runeson et al. [12]. Their approach first cleaned the textual bug reports via natural language processing techniques - tokenization, stemming and stop word 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 [6]. 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.

5. BASE LINE RESULTS

6. EVOLUTION

Until 2010, several techniques have been proposed using various similarity based metrics to detect candidate duplicate bug reports for manual verification. Automating triaging has been proved challenging as two reports of the same bug could be written in various ways. There is still much room for improvement in terms of accuracy of duplicate detection process. In [5], Sun et al. leveraged recent advances on using discriminative models for information retrieval to detect duplicate bug reports more accurately. They have validated their approach on three large software bug repositories from Firefox, Eclipse, and OpenOffice and showed that their technique could result in 17âĂŞ31%, 22âĂŞ26%, and 35âĂŞ 43% relative improvement over state-of-the-art techniques in OpenOffice, Firefox, and Eclipse datasets respectively using commonly available natural language information only.

In [5], they propose a discriminative model based approach that further improves accuracy in retrieving duplicate bug reports by up to 43% on real bug report datasets. Different from the previous approaches that rank similar bug reports based on similarity score of vector space representation, they developed a discriminative model to retrieve similar bug reports from a bug repository. They use a classifier to retrieve similar documents from a collection. Then they build a model that contrasts duplicate bug reports from non-duplicate bug reports and utilize this model to extract similar bug reports, given a query bug report under consideration. Menzies and Marcus also suggested a classification based approach to predicting the severity of bug reports.

Their basic workflow for training the discriminative model is as show below. In [2], Nguyen et al. propose BugScout, an automated approach to help developers reduce such efforts by narrowing the search space of buggy files when they are assigned to address a bug report. BugScout assumes that the textual contents of a bug report and that of its

Figure 1: Workflow

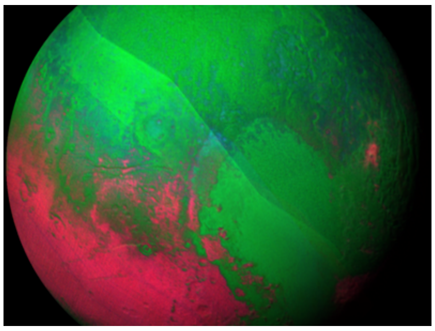

Figure 2: Accuracy with varied number of topics

corresponding source code share some technical aspects of the system which can be used for locating buggy source files given a new bug report. Instead of text based discriminative model, they developed a specialized topic model that represents technical aspects as topics in the textual contents of bug reports and source files, and correlates bug reports and corresponding buggy files via their shared topics. BugScout can recommend buggy files correctly up to 45% of the cases with a recommended ranked list of 10 files. BugScout also correlates the topics in both source files and bug reports, and uses topics as a random variable in its model which made it outperformed other approaches based on LDA+VSM.

In [3], Nguyen et al. introduced DBTM, a duplicate bug report detection approach that takes advantage of both IR-based features and topic-based features. This is different from the model discriminative model used by Sun et al. in [5]. DBTM models a bug report as a textual document describing certain technical issue(s), and models duplicate bug reports as the ones about the same technical issue(s). Trained with historical data including identified duplicate reports, it is able to learn the sets of different terms describing the same technical issues and to detect other not-yet-identified duplicate ones. Evaluation on real-world systems shows that DBTM improves the state-of-the-art approaches by up to 20% in accuracy. To support the detection of duplicate bug reports, they specifically developed a novel topic model, called T-Model.

They also used BM25F, an advanced document similarity function based on weighted word vectors of documents.

The following diagrams show the accuracy with varied number of topics: The accuracy comparison of various other algorithms and DBTM:

In [4], Sun et al. proposed a retrieval function (REP) to measure the similarity between two bug reports. It fully utilizes the information available in a bug report including not only the similarity of textual content in summary and description fields, but also similarity of non-textual fields such as product, component, version, etc. For more accurate measurement of textual similarity, they also extended BM25F used in [3] - an effective similarity formula in information retrieval community, specifically for duplicate report retrieval. Lastly they also use a two-round stochastic gradient descent to automatically optimize REP for specific bug repositories in a supervised learning manner.

Their approach is built upon BM25F model. BM25F is meant for facilitating the retrieval of documents relevant to a short query. In duplicate bug report detection, given a bug report (which is a rather lengthy query), all related bug reports need to retrieved. They also extend BM25F model to better fit duplicate bug report detection problem by extending the original model to better fit longer queries. Aside from the performance improvement, REP also significantly reduces the runtime compared to our past approach that uses SVM. The following table shows the time needed to finish an experiment run for each dataset.

<include svmVS RVP table>

For retrieval performance of REP, compared to the previous works based on SVM, it increases recall rate by 10âĂŞ27%, and MAP by 17âĂŞ23%; REP performs with recall rate of 37âĂŞ71% (1âĽď'kâĽď'20), and MAP of 46% which is considerably better than the results mentioned in [5].

In [11], Amoui et al came up with a different approach to look into the problem of duplicate bug detection. So far most of the study have been primarily on improving search and classification algorithms but the model themselves were not capable of learning. In this paper, the authors took into consideration the domain and characteristics of a software project for effective and high quality duplicate defect detection. In view of this, they needed a feedback system to assess the search quality and tune parameters for each software product and hence developed the Duplicate Defect Detection (DDD) framework. A high-level view of the framework is as shown in the following figure.

Insert <duplicatedefect.png>

DDD is composed of a customized search-engine and a feedback loop for evaluating and tuning the search quality for each project and its user requirements. In this section, we first describe the search process and then we elaborate on the parameter tuning and evaluation process of this framework.

The experimental studies allow us to analyze the impact of each tuning parameter in duplicate-defect detection performance for the selected datasets. In this study, we identify a set of parameters that are usually left out in search-based software engineering approaches, while their default values are not necessarily optimal for the input dataset. We also show that some parameters can greatly impact the search quality, but their optimal value may not be the same for all datasets and their impact is not constant. For example, among all tuning parameters, a particularly notable parameter was the Maximum Document Frequency. ItâĂŹs optimal value for BlackBerry dataset is 5% while for Mozilla Firefox is 20%.

7. NEGATIVE PATTERNS

In [5], the authors say that with big increase of recall rates, the runtime overhead of their detection algorithm is also higher than past approaches. In the largest dataset Firefox in the experiment run of their approach, the initial cost to detect a duplicate report is one second. However, as the training set continually grows and the repository gets increasingly large, the detection cost also increases over time.

The major overhead is due to the fact that we consider 54 different similarity features between reports which is way ahead of the number of features used by previous models. Consequently, SVM will need more time to build a discriminative model.

8. THREATS TO VALIDITY

In [5], experiments were only performed on 4 systems. They used the same library as used in previous tools for our reimplementation which the authors themselves said is not an efficient implementation.

9. OUR RECOMMENDATION

In [11], the authors have employed one-dimensional optimization technique for finding optimal parameter values can be extended to more advanced heuristics (e.g., Genetic Algorithms, Differential evolution and so on) as we think that some of the parameters and not mutually exclusive and hence optimizing these parameters individually might negate the improvements made on other parameters. They can also explore the search space more efficiently, and suggest pareto optimal values when optimizing for conflicting objectives (i.e., measures).

10. REFERENCES

[1] C. S. A. Nyugen, D. Lo. Duplicate bug report detection with a combination of information retrieval and topic modeling. *International Conference on Automated Software Engineering*, 2012.

[2] J. A.-K.-H. V. N. T. N. N. Anh Tuan Nguyen, Tung Thanh Nguyen. A topic-based approach for narrowing the search space of buggy files from a bug report. *IEEE Transactions on Software Engineering*, 2011.

[3] T. N. N.-D. L. C. S. Anh Tuan Nguyen, Tung Thanh Nguyen. Duplicate bug report detection with a combination of information retrieval and topic modeling. *IEEE International Conference on Automated Software Engineering*, (27).

[4] S.-C. K. J. J. Chengnian Sun, David Lo. Towards more accurate retrieval of duplicate bug reports. *IEEE Transactions on Software Engineering*, 2011.

[5] X. W. J. J. S.-C. K. Chengnian Sun, David Lo. A discriminative model approach for accurate duplicate bug report retrieval. *ACM/IEEE International Conference on Software Engineering*, 1:45 – 54, 2010.

[6] L. Hiew. Assisted detection of duplicate bug reports. *MS Computer Science Thesis, Department of Computer Science, British Columbia, Canada*, 2003.

[7] G. C. M. J. Anvik. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Transactions on Software Engineering and Methodology*, 20(3), 2011.

[8] G. C. M. J. Anvik, L. Hiew. Coping with an open bug repository. *eclipse âĂŹ05: Proceedings of the 2005 OOPSLA workshop on Eclipse technology exchange*, pages 35 – 39, 2005.

[9] Z. R. W. Z. J. Xuan, H. Jiang. Developer prioritization in bug repositories. *International Conference on Software Engineering*, (34):25 – 35, 2012.

[10] G. C. M. John Anvik, Lyndon Hiew. Who should fix this bug? *International Conference on Software Engineering*, pages 361 – 370, 2006.

[11] A. A.-D. L. T. Mehdi Amoui, Nilam Kaushik. Search-based duplicate defect detection: An industrial experience. *IEEE Transactions on Software Engineering*, 40(99):173 – 183, 2013.

[12] O. N. P. Runeson, M. Alexandersson. Automatic summarization of bug reports. *International Conference on Software Engineering*, 2007.

[13] G. M. Sarah Rastkar, G Murphy. Automatic summarization of bug reports. *IEEE Transactions on Software Engineering*, 40(99):366 – 380, 2014.

[14] e. a. Sun, Chengnian. A discriminative model approach for accurate duplicate bug report retrieval. *IEEE International Conference on Software Engineering*, 1(32), 2010.

[15] P. J. Sureka, Ashish. Detecting duplicate bug report using character n-gram-based features. *Software Engineering Conference (APSEC)*, 2010.

[16] H. Y.-R. Z. Z. W. L. Z. W. X. Xuan J, Jiang H. Towards effective bug triage with software data reduction techniques. *IEEE Transactions on Knowledge and Data Engineering*, (27):264 – 280, 2015.