A Survey on Automated Duplicate Detection in a Bug Repository

Nawagata Nilambari
PG Scholar, Computer Science and Engineering,
Galgotias University, Greater Noida,
U.P, India

Vintee Chaudhary
PG Scholar, Computer Science and Engineering,
Galgotias University, Greater Noida,
U.P, India

Shivani Gautam
PG Scholar, Computer Science and Engineering,
Galgotias University, Greater Noida,
U.P, India

***Abstract*— in a bug repository whenever a new bug is reported it is important to analyze this new bug report in order to detect it as duplicate or non duplicate. For this purpose plenty of work has been done in this area. In this paper we go through the previously proposed techniques and analyze the contribution of each work in making duplicate detection in bug repository such an important area of research.**

Keywords—duplicate; bug report

I. INTRODUCTION

In software maintenance phase, the issue's that the user encounters is reported to the developer via a bug report. A bug report consists of certain details like the *product, version, component, severity, priority, title, summary, description* associated with the bug. The quality of the bug report depends on the knowledge of the user. An experienced user is more likely to report the bug with appropriate information than an inexperienced user who might not be well known to the bug that has arisen. The problem occurs when multiple users send the same bug report to the developer but details provided in the bug report differs in term of description given for the bug. Now it is a challenge for the developer to analyze the incoming bug report and mark them as duplicate of existing bug report or to mark them as non duplicate. For this purpose s(he) need to give lots of time to manually detect whether an incoming report is duplicate or not. 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" [1], this explains the need of an automated system for duplicate detection in a bug repository. In one of the approach [4] the duplicates were considered useful as they add information to the same pre-existing bug report. In other approaches the duplicates were treated differently [5-7]. We would therefore explain the bug, its life cycle, and the various fields of a bug report and the various approaches for duplicate detection in the subsequent sections.

II. BACKGROUND

A. *What is a bug?*

A bug is an error, mistake, flaw, failure or fault that occurs in a computer program and restricts the proper functioning thereafter. These bugs may arise due to gaffe in the design or the code of the program. As a result of which the program stops abruptly or gives inaccurate results.

B. *Life cycle of a bug*

The life cycle of a bug can be summarized in following phases:

1. New: when a new bug is encountered and is reported to the developer.
2. Assigned: the bug in this phase is assigned to the developer for further analysis.
3. Resolved: the bug has been resolved by the developer in this phase and is assigned a status as fixed, duplicate, won't fix.
4. Verified: the test result provided by the developer is verified by the tester in this phase.
5. Closed: the bug is finally closed after an appropriate solution is provided to it.
6. Reopened: if the quality assurance team is not satisfied by the solution provided for the bug, it is reopened.

C. *Fields in a bug report*

The important fields in a bug report can be broadly classified into two categories based on the nature of portrayal:

1. Textual Features: these are natural language text it includes summary and description of the bug.
2. Categorical Features: these are features in non textual format; it includes product, component, version, priority, type.

Table I shows the important fields in a bug report.

TABLE I:
FIELDS OF INTEREST IN A BUG REPORT

Field	Description
Summ	*Summary*: concise description of the issue
Desc	*Description*: detailed outline of the issue, such as what is the issue and how it happens
Prod	*Product*: which product the issue is about
Comp	*Component*: which component the issue is about
Vers	*Version*: the version of the product the issue is about
Prio	*Priority*: the priority of the report, *i.e.*, *P1*, *P2*, *P3*, $\cdots$
Type	*Type*: the type of the report, *i.e.*, *defect*, *task*, *feature*

III. MOVITATING EXAMPLES

In past several years the open source bug repository such as Mozilla, Firefox, and Eclipse have encountered numerous duplicates. There are examples which show how reports can be duplicates of each other.

For instance considering the bug repository of Mozilla, the body of bug report #464484, submitted on September 22, 2010, includes the text "provide writeable interface to update add-on .xpis "and the bug report #586703 reported on August 12, 2010 has text "API put the U in CRUD for the Add-on model". These two bug reports were marked as duplicate of each other.

Similarly, the bug report #534675, reported on December 14, 2009, includes text "update info escapes all tags, messes up markup" and the bug report #540968 has text "back out and fix ugly update info" which was reported on January 20, 2010 were marked duplicate of each other.

Another example is of bug report #541074 reported on January 21, 2010 which has text "Firefox extension update from beta channel fails to download" and bug report #537720 which was submitted on January 4, 2010 had text "Beta channel add-on updates are missing from http://releases.mozilla.org server" were similarly marked duplicate.

The above example shows that the bugs that have been reported are actually duplicates of each other but they differ in textual representation. The words "add-on", "update", "info", "beta", "channel" are used repeatedly in the bug reports and the similarity among them are evident.

These examples and many other similar instances are the motivation behind the need of a tool that can automatically detect duplicates in a bug repository so as to save the time and efforts of the triagers.

IV. METHODOLOGY

A. *Approach for duplicate detection*

There can be two approaches for duplicate detection; one can be to restrict the duplicates from reaching the developer by filtering it as it arrives. This can be done by categorizing the new incoming bug report as duplicate or non-duplicate.

Another approach is to return the top k-most similar bug report based on the features that are similar between the query report and the rest of the repository. This approach returns the list of top related bug reports.

Both of the above mentioned approaches have been worked upon. Jalbert and Weimer [3] have worked on the first approach and the experiment conducted was able to eliminate 8% of duplicate reports. Sun et al. [5-7] have focused on the second approach and their experiment has showed relevant improvement over the first approach.

There are related works that do not consider bug reports to be harmful and propose to combine the duplicate bug report with the master bug report. By doing so the information associated with the bug report gets enhanced. One such work has been done by N. Bettenburg et al. [4].

B. *Method for duplicate detection*

There is basically one method employed for detecting duplicates in a bug repository by using natural language processing.

This can be combined with execution traces and categorical information associated with the bug report.

The Natural Language Processing is done by analyzing the textual attributes of a bug report such as Title, Summary and Description. There are some of the research works [2, 3] where only Natural Language Processing is used.

Some research works [5, 6] have included both the categorical attributes (that includes Type, Product, Component associated with the bug) as well as Natural Texts attributes for better performance.

Some research work as that of Wang et al. [8] has used added information of execution traces along with Natural Language Processing to detect the similarity. But their study considered only a small subset of bug repository due to the fact that execution traces are scarcely associated with the bug reports.

It can thus be seen that Natural Language Processing is an important aspect of duplicate detection.

C. *Comparision Technique*

Almost all of the studies take the benefit of the Information Retrieval system and each try and put their effort in improving the previous state-of-art.
The comparison technique adopted can be briefly categorized as:

- Applying vector space model and cosine similarity metrics to detect the similarity.
- Applying SVM to predict duplicates.
- Comparing the reports textually using TF-IDF and cosine similarity metrics.

The above mentioned are general techniques that have been implemented.
One of the different approaches is that proposed by Sureka et al. [9] that uses the unigrams of characters in the Description for comparison.
Another is that done by Sun et al. [6], the proposed system uses BM25Fext for similarity measurement along with 7 features comparison to train their model

V. OVERALL FRAMEWORK

Fig.1. depicts the overall framework towards duplicate detection. It has been explained in the subsequent subsections.

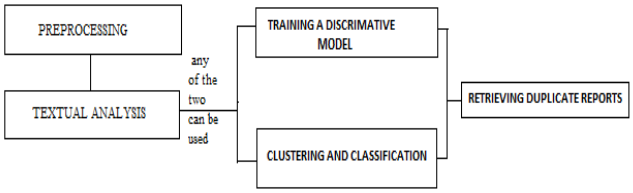

Fig.1. : Shows general framework for duplicate retieval

A. Step 1: Data Preprocessing

Data pre-processing is the initial step towards duplicate detection using Natural Language processing techniques. In this stage the data is processed prior to its use. For this purpose following steps are taken:

1) Tokenization: in tokenization the document is parsed and divided into tokens by removing the delimiters (spaces, puntuation marks).

2) Stemming: in stemming the words are reduced to their ground forms, for example a stemmer will reduce 'runs', 'running' to its ground form 'run'.

3) Stop word removal: in this step the stop words are removed from the data by eliminating words that are of least importance, such words are 'the', 'and', 'is' and many more.

B. Step 2: Textual Analysis

1. Word weighing function

The summary and description of a bug report is converted into their equivalent weight vectors with the help of TF-IDF, it is the most popular function for weighing the words. TF stands for Text Frequency. It corresponds to the number of times a term occur within the document, as in (1).

$$tf(t,d) = 0.5 + \frac{0.5 * f(t.d)}{max\{f(w,d) : w \in d\}} \quad (1)$$

IDF stands for Inverse Document Frequency. It is the contextual measure of the importance of a word. It works on the assumption that important words will occur frequently in some document and infrequently across the entire corpus, this can be seen in (2), which is the standard formula for IDF.

$$idf(t,D) = log\frac{|D|}{|\{d : D : t \in d\}|} \quad (2)$$

And then the TF-IDF weighing factor is calculated as (3), this is one of the most popular word weighing factor which forms word-vector out of text.

$$tf - ifd(t,d,D) = tf(t,d) * idf(t,D) \quad (3)$$

2. Similarity measurement function

After the weight vectors have been calculated for each word in a document, the similarity among the words are calculated by using similarity measurement function. The most popular similarity measurement function is Cosine Similarity as shown in (4), it is the most widely used metrics for similarity measurement and it has been used in several research works [3, 6].

$$similarity = \cos(\theta) = \frac{v_1 \bullet v_2}{|v_1| \times |v_2|} \quad (4)$$

Conversely, a distance measurement function like Jaccard and Dice can also be used to measure the distance between two similar words. It has been used by Runeson et al. [2] to measure the similarity.

Another retrieval function is BM25F. Sun et al. [6] extended BM25F as in (5). BM25F is used for processing queries with shorter length; they extended it so as to process queries with longer length which is advancement to the previous state-of-art.

$$BM25F_{ext}(d,q) = \sum_{t \in d \cap q} IDF(t) \times \frac{TF_D(d,t)}{k_1 + TF_D(d,t)} \times W_Q \quad (5)$$

$$\text{where } W_Q = \frac{(k_3+1) \times TF_Q(q,t)}{k_3 + TF_Q(q,t)}$$

C. Step 3: Training model by SVM or Classification and Clustering

In this step either of the two method of identifying the duplicate can be implemented i.e.Training model via SVM or Classification and Clustering.

1) Training model via SVM: SVM is a powerful machine learning technique. On the basis of a set of labeled vectors, it builds a classifier that classifies the vectors into positive and negative class, separated by a hyper-plane as shown in fig.2. This predicted model can be further used to classify other unknown data points. A popular implementation of SVM is libsvm [10], which is used in some of the research works [5-7] that uses SVM for prediction and classification.

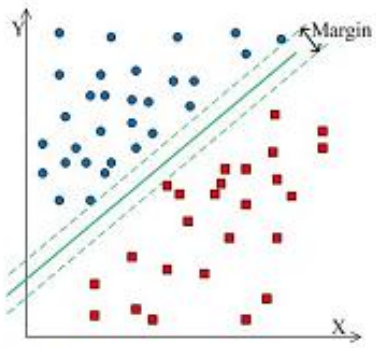

Fig.2. : Shows maximum margin hyperplane seperating two classes as calculated by SVM

2) Classification and Clustering: for clustering and classification a graph clustering algorithm is implemented. In the graph the nodes represent defect report and the edges link reports with similar text. The most popular algorithm for clustering is Mishra et al. [11] which produce a set of possibly-overlapping clusters given a graph with un-weighted, undirected edges.

The clusters thus formed have high coherence within the cluster and low coherence among the clusters.

D. *Step 4 : Retrieving duplicate report*

The retrieval procedure can give two outcomes based on the method implemented (Section IV A).

- One of the outcomes may be a binary answer (yes/no) to the new incoming report (query) whether it belongs to the duplicate class or non duplicate class.
- Another outcome may be a list of k-top similar bug reports that were found to be similar to the new incoming bug report by the trained model.

E. *Performance Measurement:*

To analyze the performance of the proposed model or method, generally two parameters are used:

a) Recall Rate: as in (6), the recall rate is the measures the fraction of duplicate reports whose master has successfully been detected among the total reports.

$$recall\ rate@k = \frac{N_{detected}}{N_{total}} \quad (6)$$

b) Mean Average Precision (MAP): the approach where a query can have multiple relevant documents, MAP is used to measure the rank of retrieval result (7) is the reduced form of MAP.

$$MAP(Q) = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{index_i} \quad (7)$$

VI. OTHER REALTED RESEARCH

A. *Duplicate detection related studies:*

There are some other contributions in this field; some of them are an improvement over the existing state-of-art while others are a completely different approach, such as the work by Nguyen et al. [12] they proposed a new technique DBTM that detects duplicate by combining IR and Topic Modeling. To extract the topics from the bug reports they proposed LDA-based technique called T-Model. The words in the bug report and the duplication relation among them is used to estimate the topic. The dataset was from open sources Mozilla, Eclipse and Open Office, and an improvement of 20% was reported over the state-of-art.

Another approach for duplicate detection was the consideration of Contextual features apart from the textual and categorical features, by Alipour et al. [13]; in this research they extended the work of Sun et al [6] by including contextual features and the similarity was measured by using BM25F. Contextual data-sets included software architectural words, nonfunctional requirement words, topic words extracted by LDA, topic words extracted by Labeled-LDA, and random English words. For duplicate retrieval, a pair of data set containing contextual, textual and categorical features in a bug report was fed to the machine learning classifier to classify the new bug report as duplicate or non duplicate and the result showed an improvement of 11.5% over the work done by Sun et al. [6].

B. *Other bug report related studies:*

Plenty of work has been done in concern with the bug reports some of them are as follows:

Categorizing the bug report has been studied by Anvik et al. [1], Cubranic and Murphy [14], Pordguski et al. [15] and Francis et al [16], these paper focused on assignment of the bug report to the right developer. A study on the severity of the bug report was carried by Menzies and Marcus [17]. Hooimeijer and Weimer [18] developed a model that could predict the quality of a bug report. Sandusky et al. studied the statistics of duplicate bug reports [19]. Bettenburg et al. studied the developers of Eclipse, Mozilla, and Apache to analyze what developers care most in a bug report [20].

VII. CONCLUSION

The purpose of this survey is to provide an overview of the process that is followed from the scratch towards the retrieval of duplicates in a bug repository. There can be several approaches, so this paper primarily focused on the basic steps that are followed in general for duplicate detection, and gave a glimpse of how actually things are being worked out.

REFERENCES

[1] J. Anvik, L. Hiew, and G. C. Murphy, "Coping with an open bug repository," in eclipse '05: Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange, 2005, pp. 35–39

[2] 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

[3] N. Jalbert and W. Weimer, "Automated Duplicate Detection for Bug Tracking Systems," in proceedings of the International Conference on Dependable Systems and Networks, 2008

[4] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicate bug reports considered harmful ... really?" in ICSM08: Proceedings of IEEE International Conference on Software Maintenance, 2008, pp. 337–345

[5] 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, pp. 45–56

[6] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "Towards More Accurate Retrieval of Duplicate Bug Reports," in ASE, 2011

[7] Y. Tian, C. Sun, and D. Lo, "Improved Duplicate Bug Report Identification," in proceedings of 16th European Conference on Software Maintenance and Reengineering, 2012

[8] 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

[9] A. Sureka and P. Jalote., "Detecting duplicate bug report using character n-gram-based features," In the proceedings of Software Engineering Conference (APSEC), 2010 17th Asia Pacific, pages 366–374. IEEE, 2010

[10] C.-C. Chang and C.-J. Lin. LIBSVM: a library for support vector machines,2001.Available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[11] N. Mishra, R. Schreiber, I. Stanton, and R. E. Tarjan. Clustering social networks. In Workshop on Algorithms and Models for the Web-Graph (WAW2007),pages56–67,2007

[12] A. T. Nguyen, D. Lo, C. Sun, "Duplicate Bug Report Detection with a Combination of Information Retrieval and Topic Modeling," in ASE 2012

[13] A. Alipour, A. Hindle, and E. Stroulia. A contextual approach towards more accurate duplicate bug report detection. In Proceedings of the Tenth International Workshop on Mining Software Repositories, pages 183–192.IEEEPress,2013

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

[15] A. Podgurski, D. Leon, P. Francis, W. Masri, M. Minch, J. Sun, and B. Wang, "Automated support for classifying software failure reports," in Proceedings of the 25th International Conference on Software Engineering, 2003, pp. 465–475.

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

[17] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in ICSM08: Proceedings of IEEE International Conference on Software Maintenance, 2008, pp. 346–355

[18] 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.

[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 International Workshop on Mining Software Repositories, 2004, pp. 80–84.

[20] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "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.