Duplicate Bug Detection

Bhanu Anand
Computer Science Department
North Carolina State University
Raleigh, NC, 27606, USA
bhanua@ncsu.edu

Esha Sharma
Computer Science Department
North Carolina State University
Raleigh, NC, 27606, USA
esharma2@ncsu.edu

Vinay Todalge
Computer Science Department
North Carolina State University
Raleigh, NC, 27606, USA
vntodalg@ncsu.edu

***Abstract* – Very large software systems evolve consistently throughout the lifetime of the system. This leads to continuous reporting of bugs which need to be resolved. Many times, the same issues get reported more than once. Also, two bugs which essentially have the same issues may be reported separately claiming that they are the different. There are multiple reasons for duplicate bugs to exist. Maintaining such kind of bug repository is tedious and cumbersome because of the amount of maintenance and supervision it entails. One option is to automate this job of detecting duplicate bugs wherein triaging can be done for the same. Many studies have been conducted on areas related to this. These studies include studies on areas like assignment of duplicate bugs to specific developers, using duplicate bug reports to generate relevant data to resolve bugs and using duplicate bug reports to predict bugs which may come up which turn out to be duplicates. In this study, we try and summarize the work done in this area. All the papers studies by us are related to the duplicate bug detection domain. We also comment on the improvements/issues in those papers and possible future improvements.**

Index Terms – Duplicate Bugs, Triaging, Bug Repositories, Developer Recommendation, top-k similar reports, De-duplication.

INTRODUCTION

Given the current rate of technology evolution, software is ubiquitously used. Due to the high level of complexity of tasks performed by a software and the multitude of tasks performed by the same software, these softwares become humungous and very cumbersome to manage. Maintenance of such a software system is an especially difficult task. Also, with time, software systems evolve. Evolution of a system means adding new requirements, enhancing existing requirements and removal of bugs. With evolution of a system, bugs get added to the system. In this study, we concentrate on how these bugs can be detected and how this process impacts the maintenance of the system.

Bugs can be defined as incapability of code to deliver promised functionality. Because of bugs in a software the software may not be able to deliver the behavior expected from it. Some of the reasons these bugs come up are bad design of the software, careless development practices, incomplete analysis of requirements and insufficient or improper [1], [2]. Most companies have their own tools for reporting bugs. There are tools which are commercially available as well maintaining bugs like Bugzilla. Bugzilla is a web-based general-purpose bug tracking and testing tool originally developed and used by the Mozilla project, and is licensed under the Mozilla Public License.

Reporting of bugs sometimes depends upon cultural and/or company environments. Reporting a bug is dependent upon of habitual characteristic of person who is reporting a bug. Bugs can be reported by developers, testers, designers, managers etc. Bugs can be directly reported in bug tracking tools like Bugzilla or can be documented and be added into bug repositories periodically. Most of bugs are reported in the testing phase. Bugs reported are then analyzed and assigned to appropriate developers/resources for corrective actions. After a given bug is fixed, the bug is made inactive.

There is a likely chance that the same bug may be reported more than once. Some reasons for this can be the lack of motivation in users to use bug tracking tools correctly and defects in the search engine of the bug-tracking systems [3]. More than one bug report can report the same bug behavior although in different ways. The same bug can also be reported in different contexts or in different environments. The problem with undetected duplicate bugs is that the same bug may be assigned to more than one developer and hence there will be more than one resource addressing the same issue. This is a waste of resources.

When a bug is reported in bug repository, it is scanned through by a Triager. Triager is person who decides whether a reported bug is duplicate or not. Each reported bug is checked against all present bugs in repository. If the reported bug is found to be a duplicate, then it is marked as duplicate and handled accordingly. Else, it is added to the bug repository and assigned to an appropriate developer after bug analysis. Manually triaging however takes a considerable amount of time and its results are unlikely to be accurate. For example, Mozilla in 2005 reported that on an average everyday there are almost 300 bugs appear that need triaging. This was reported to be far too much for Mozilla programmers to handle [5]. This could lead to inefficient use of resources. Considering the large number of bugs reported daily and the amount of work needed to be put in by the Triager, there have been a number of studies carried out in order to automate process of tedious duplicate bug report detection [4].

There is a process to analyze a bug and assign it to an appropriate developer. This process can be automated as well. The automation can be done in such a way that if we have sufficient knowledge about a bug we can find a developer to work on it in advance.
The second important aspect to be noted is that though duplicate bugs mostly carry redundant data, they are not necessarily useless. There may be findings reported from duplicates that can help us find different aspects of bugs. Duplicate bugs could give us more insights into a functionality that may not be working. If many bugs are reported on the same functionality it necessarily means there is something which needs to be corrected in that functionality. Bettenburg et al. stated that one report can give us only one view of problem, but duplicate bug reports can complement each other [3].

Bugs reports can be compared against one another and conclusions can be drawn from there about duplicity. Most common form for doing this comparison is to compare their textual representations. Sometimes bug reports may be ranked and this ranking is compared to check to what extent two given reports are similar to each other. Studies in [6], [7] and [8] do just this. They process natural language text of bug reports to produce a ranking of related bug reports.

There is one more approach to address duplicate bugs. Instead of marking reported bug as duplicate, one can maintain a bucket of bugs having one master report and other slaves as duplicates. Top-n related bugs can be found for each new bug reported and made master and if a reported bug is marked as duplicate then it can be assigned to that bucket [6]-[8] and be made its slave.

MOTIVATION

As software systems grow in size, they become more cumbersome to manage and code repositories become difficult to maintain. This adds up to maintenance work. Maintenance activities account for over two thirds of the life cycle cost of a software system, and are reported to sum up to a total of $70 billion per year in the United States [9]. Software maintenance is critical to software dependability, and defect reporting is critical to modern software maintenance. Considering strict guidelines for budgets in software companies, significant expenditure of money on maintaining bug repositories is not considered as a wise distribution of resources. In case of duplicates, more than one developer can end up working on fixing the same issue. The same problem applies to testers, testers testing the same issue at the same time lead to unnecessary wastage of resources. In addition to all of this, there are no robust techniques for comparing two bug reports in order check for duplicates. Most obvious technique would be comparing textual representations of two given bug reports [10]. This includes finding similarities between 2 reports by making use of similarity in words, fields in case of reports and technical words specific to domains. But again, these features cannot detect the duplicity of reports with accuracy. It involves processes like removal of stopping words, getting rid of noisy data in a report which need to be done as clean up. Overall accuracy in finding duplicates depends not only on the algorithm used for finding textual similarity, but also how effectively the initial clean-up of reports is done.

This motivates the idea of more robust and accurate automation of bug triaging, where most of work involved is automated and leads to more accurate finding duplicates saving significant amount of resources.

The motivation for this study is to study the various papers associated with finding duplicate bugs and cleaning data and summarize how they have reported to be doing the same.

HYPOTHESIS

In the papers studied for this report, it is hypothesized that finding an accurate and stable approach for finding duplicates and automating this process can improve overall project lifecycle management system. In some of the papers studied, some algorithms are also proposed. It is predicted that by applying these algorithms, assuming appropriate accuracy in them, most bugs can be assigned automatically to developers to work further. It was also proposed that there is large chance of discovering hidden and/or obscure details of bugs from duplicate bugs, as sometimes duplicate bugs compliments each other. To summarize, if one gets enough insights for a particular bug using most the approach which is most appropriate, an attempt can be carried out to determine root cause of bug

RELATED WORK

Information Retrieval is the method to extract information from unstructured documents, most of which are expressed in natural language. Generally bug reports are in the form of natural language. Hence these two fields are related to each other. We will cover related work in Information Retrieval and bug deduplication.

A. *Information Retrieval in Software Engineering*

Hindle et al. implemented a topic extraction method based on texts called as labels [11]. They have used log repositories for LDA topic extraction. Using non-functional requirements concepts, authors believe that these technique are cross platform independent of systems.

Latent Dirichlet Allocation (LCD) is a new model for maintaining relations where each document is related to a topic, used by Blei et al [12]. They have used convexity-based variation approach for inference and demonstrated that this is a fast algorithm which gives reasonable performance to detect the relationships between documents.

Marcus et al. used latent semantic indexing (LSI), a generative probabilistic model for sets of discrete data, for linking explained queries with corresponding code snippet [13]. This concept can be further modified for modularizing software systems.

B. *Bug Report Deduplication*
Study by Just et al. shows that there have been issues with interface of the bug tracking software itself, which causes less precision while reporting bugs [13]. There might be some of the fields which are not available while a bug report is being filed. In this case, the user may get confused and this could lead to unpredictable data inputs. Trying to submit all of the observations in restricted fields due to unavailability of fields while reporting bug reports, can cause addition of irrelevant data in certain fields.

In first paper [17] we read, Sun et al. has proposed a new retrieval function (REP) for finding textual similarity between two bug reports. For an accurate measurement of textual similarity, they have extended BM25F – an effective similarity formula in the information retrieval community. It fully utilizes the similarity in the information available in a bug report which includes the similarity not only in the textual content in the summary and description fields, but also the similarity in the non-textual fields such as product, component, version, etc. Using these supplementary fields adds more relevance when we are trying to find similarity. On 209058 reports from Eclipse, authors claims that were able to achieve a recall rate of 37-71% and mean average precision rate of 47%. They have followed an approach where in whenever a new bug is reported, top-k similar bug reports are retrieved which utilizes Betternburg's et al.'s idea that multiple bug reports complements each other.

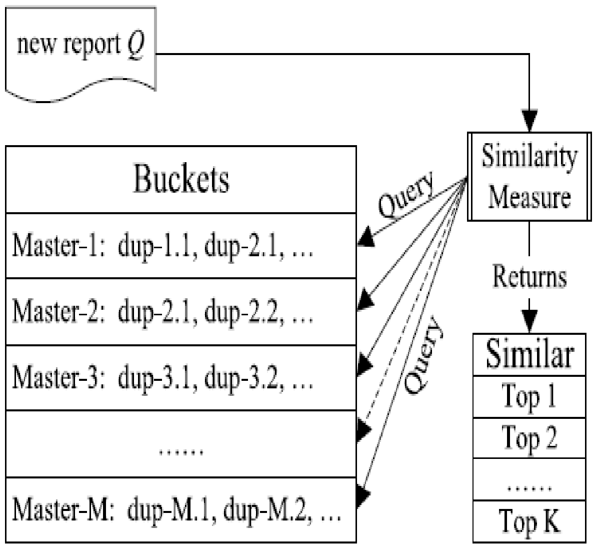

new report Q
Similarity Measure
Buckets
Query
Master-1: dup-1.1, dup-2.1, ...
Master-2: dup-2.1, dup-2.2, ...
Master-3: dup-3.1, dup-3.2, ...
......
Master-M: dup-M.1, dup-M.2, ...
Query
Returns
Similar
Top 1
Top 2
......
Top K

Fig. 1. Overall Workflow for Retrieving Duplicates

In above figure, also picked from paper 1 [17] authors use queries in the bug repository in order to retrieve top-k similar reports which complement each other. Using similarity measures, each bug reported is mapped to other queries which find similar bugs. Each new bug report is queried using similarity measures against all buckets found in bug repository. These queries would return top-k related bug reports. These top-k related bug reports would be analyzed further for deciding duplicity of bug report.

In the second paper we studied [18], Jalbert et al. have proposed an automation system which uses surface features [14], textual similarity metrics [15] and graph clustering algorithm [16] to identify duplicate bugs. Authors have used 29000 bug reports from Mozilla having 25.9 % of duplicate bugs. They were able to reduce development cost by filtering out 8% of duplicate bug reports while allowing at least one report reporting at least one real defect to reach developers. For finding textual similarity, they have used Inverse Document Frequency (IDF) which is based on principle that important words will appear frequently in some documents and infrequently across the whole of the corpus. [18] Defines IDF as:

$$\mathsf{IDF}(w) = \log\left(\frac{\text{total documents}}{\text{documents in which word } w \text{ appears}}\right)$$

Surprisingly authors found that employing IDF to find duplicates resulted in strictly worse performance than a baseline study where in pnly non-contextual term frequency was used.

This study concluded that this techinique did not work because word frequency should not be the sole reason in comparision of duplicates. They elaborate on this reason by taking the example of two totally different bug report from same domain. In this case, as these two bug reports are from same domain, their contextual information would likely be the same. If we consider their's textual representation, there is high chance finding same words and/or technical terms refering to some actions from same domain. If we consider term frequency of those feature words from both bug reports, there might be little variation found in terms of numbers. But there will not be any significant differences in term frequencies because for explaining certain tasks from same domain, certain words would be repeated to describe the action.

To determine the performance of the algorithm used , the authors used the TF/IDF algorithm as baseline . This algorithm uses the TF/IDF weighting that takes inverse document frequency into account; it is a popular default choice. Authors have considered the Runeson et al. algorithm with and without stoplist word filtering for common words. Using stoplists improved performance of the algorithm. It is interesting to note that while the algorithm is strictly worse at this task than either algorithm using stoplisting, the TF/IDF gives better performance than the approach without stoplisting. Thus, an inverse document frequency approach might be useful in situations in which it is infeasible to create a stop word list.
TF/IDF approach underperforms other approches. Hence, authors have dismissed this approch and headed towards weightage based approach where in each appearing term in report is given weigh and updated on timely manner.

In the third paper we studied [19], Wang et al. have gone one step ahead and included execution information as well to determine duplicates. Authors found that only one of the two : Natural Language processing and Execution Information cannot be used to find similarity efficiently among bugs. They came up with some techiniques which ranks/weights the potential duplicates with combination of both approaches. In Basic Heuristics, combined similarity is just average of NLP similarity and Execution information similarity. In Classification Based Heuristics, classes/ranks were assigned to bugs according to value of NLP similarity and Execution information similarity.

In the fourth paper we studied, Betternburg et al. came up with very interesting proposal claiming that each duplicate is not necessarily just a redundant piece of information. He proposed that duplicates can compliment each other and provide very crucial information about getting more insight into a specific bug. The authors proposed an approach where in one can maintain master report and extended master reports which serves as information items which can be used while retrieving more relevant infotmation. Authors identified some of the potential reasons behind bug reporting are:

- Lazy and inexperience users.
- Poor search feature
- Multiple failures, one defect
- Intentional resubmission
- Accidentally resubmission

Authors had the following observations as well:

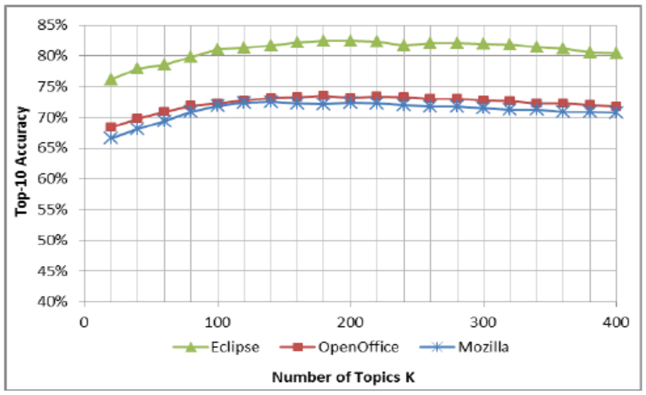
- For master reports, duplicates are often submitted by the same users.
- Often duplicates are submitted at the same time (by the same users).
- Some bug reports are exact copies of each other, but not always marked as duplicates.

In the fifth paper we studied , Sun et al. concentrated on finding the various similarity features which are distinct properties of a particular bug report. They found 54 such distinct similarity features and used discriminative model to automatically assign weights to each feature to discriminate duplicate reports from nonduplicate ones. This process is rigorous and has theoretical support from machine learning area. The author claimed that this approach is more adaptive, robust and automated because in this approch the relative importance of each feature will be automatically determined by the model through assignment of an optimum weight. Consequently, as bug repository evolves, their discriminative model also evolves to guarantee that all the weights remain optimum at all time. Authors have rigourously tested this approach on variety of data namely OpenOffice, Firefix and Eclipse and concluded that this technique would result in 17–31%, 22–26%, and 35–43% improvement over state-ofthe-art techniques in OpenOffice, Firefox, and Eclipse datasets respectively using commonly available natural language information alone. For selecting the similarity features, authors used Fisher's score having idf-based formulae. They have calculated the Fisher score of each of the 54 features using idf, tf, and tf $*$ idf. The results on Eclipse dataset shows that idf-based formulas outperform tf-based and (tf $*$ idf)-based formulas in terms of Fisher score. This lead them to select idf based formulae as feature scores.

In the sixth paper we studied , Xia et al. went one more step ahead and introduce a noval idea of automatically assignment of duplicate bugs to corresponding developer. They developed a tool called as DevRec which automatically processes a new bug report and recommends a list of developers that are likely to resolve it. DevRec takes advantage of both BR based Analysis where for a given bug report, corresponding duplicate bug reports were discovered and appropriate developers are found based on developers of past duplicates found for given bug and D based Analysis where affinity/distance of a developer towards given bug report is calculated and appropriate developer is assigned considering features of bug reports and characteristic of old bug that a specific developer has fixed. DevRec (new tool introduced in this paper) improves the average recall scores of Bugzie by 57.55% and 39.39%, in two separate categories. DevRec also outperforms DREX which is an existing tool used by author as baseline for comparison by improving the average recall scores by 165.38% and 89.36% in two separate categories.

In the seventh paper we studied , Nguyen et al. introduced DBTM, a duplicate bug report detection approach that takes advantage of both Information Retrieval -based features and topic-based features. 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).It then trains the algorithm with historical data including identified duplicate reports. Through this approach the algorithm was able to learn the sets of different terms describing the same technical issues and to detect other not-yet-identified duplicate issues reported.

The figure shown below(also picked from [20]) shows the work of the author in comparing the sensitivity of DBTM's accuracy with respect to different numbers of topics K. They ran DBTM on 3 different bug repositories from Eclipse, OpenOffice and Mozilla. They ran DBTM on Eclipse data set as K was varied from 20 to 400 with the step of 20, and then measured top-10 detection accuracy. The shapes of the graphs for three systems are consistent. That is, as K is small (K<60), accuracy is low. This is reasonable because the number of features for bug reports is too small to distinguish their technical functions, thus, there are many



documents classified into the same topic group even though they contain different technical topics. When the number of topics increases, accuracy increases as well and becomes stable at some ranges. The stable ranges are slightly different for different projects, however, they are large: K=[140-320] for Eclipse, K=[120-300] for OpenOffice, and K=[100-240] for Mozilla. This suggests that in any value of K in this range for each project gives high, stable accuracy. The reason might be because the number of topics in these ranges reflect well the numbers of technical issues in those bug reports. However, as K is larger (K>380), accuracy starts decreasing because the nuanced topics appear and topics may begin to overlap semantically with each other. It causes the document to have many topics with similar proportions. This overfitting problem degrades accuracy.

In this approach, Each aspect/feature of software/system is considered as Topic, represented by words or terms. Bug reports are those reports which shows wrong implementation of such topics. Terms/words used in topics are maintained in one Vocabulary (Voc). Authors have used LDA (A Generative Machine Learning Model), which analyses a bug report as an instance of machine state. LDA can be trained on timely basic with the help of vocabulary maintained with model. DBTM will be trained periodically by both historical data including bug reports and their duplication information. Term frequency and duplicacy inrelation among those reports, will be used to determine topic proportion of the shared technical issue(s), and the local topic proportions of the bug reports. Authors have used again same BM25F function for retrieval of structured documents which are composed of several fields. BM25F computes the similarity between a query and a document based on the common words that are shared between them. BM25F is modified to consider word importance globally and locally.

In the eighth paper we studied Alipour et al. has proposed an approach to consider contextual information as well while determining duplicates. They investigate how contextual information, based on prior knowledge of software quality, software architecture, and system-development (LDA) topics, can be exploited to improve bug-deduplication. They have demonstrated the effectiveness of their contextual bug-de-duplication method on the bug repository of the Android ecosystem.Based on this experience, we conclude that researchers should not ignore the context of software engineering when using IR tools for deduplication. Also, in this study the authors have paid enough attention to initial clean up of overall information. As a part of initial clean up , each report is preprocessed to remove stop . Preprocessed bug reports with the help of contextual similarity were analyzed for finding out whether given bug is duplicate or not. Textual measures and contextual measure were used before machine learning algorithm is applied. Results of this paper are compared with results of [17] . Author claims that contextual approach improves accuracy in detecting duplicate bugs up to 11.55%. Reason for selecting [17] for comparing results is first paper uses textual similarity very effectively for duplicate detection. Author have wanted to compare results with model which makes use of textual similarity of bug reports very efficiently.

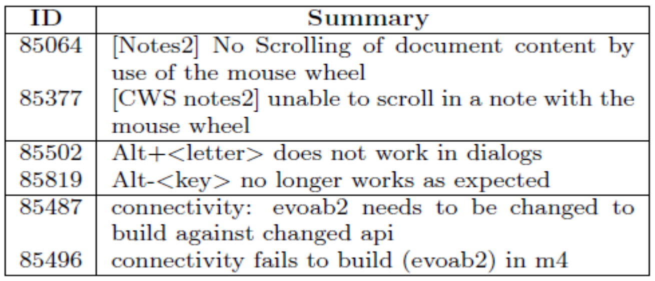
CHECKLISTS

Checklists are minimal datasets and corresponding actions considered in design analysis. As in this domain, our main aim is find whether newly reported bug is duplicate of any bug present in bug repository, most important data sets to consider is textual representation of bugs. Traditionally and conveniently bug are reported in textual formats.

Here corresponding actions would be preliminary initial procedure where given textual representation of bugs is cleaned and prepared for further analysis. These actions includes stemming (removing 'ing', 'ed' and such data), removal of stop words (a, an, the), removal of noisy data (introductions, headers and footers)

Checklist manifestos take care that relevant data is not missed for analysis. This might include checking variety of bug reports in system, bug filing practices, variety of fields used while filing a bug report, creating datasets for tuning models, periodic updates on complimentary information items used.

DATA



ID	Summary
85064	[Notes2] No Scrolling of document content by use of the mouse wheel
85377	[CWS notes2] unable to scroll in a note with the mouse wheel
85502	Alt+<letter> does not work in dialogs
85819	Alt-<key> no longer works as expected
85487	connectivity: evoab2 needs to be changed to build against changed api
85496	connectivity fails to build (evoab2) in m4

This (above figure picked from paper 5 [7]) is very typical example of how duplicate bugs looks like. We can see, thought textual representation of these bug look different, they convey the same message. If we scan above bug reports from contextual perspective, it can be seen obviously that all duplicate bug reports carry similar contextual information. For example, bug reports having ID 85487 and 85496 were reported in same domain of 'connectivity' and conveys similar issues about 'evoab2'. In addition to that, these duplicates includes similar technical terms in reports. For example, terms 'scroll' is common in bug report 85064 and bug report 85377, terms 'Alt' is present in bug report 85502 and bug report 85819.

All the papers we studied in this domain, has bug reports looks like the one shown above. Bug repositories having such bug reports distinguished by their unique ids

Most of the papers studied in this paper have used bug repositories from either of Mozilla, Eclipse, or OpenOffice. One obvious reason could be they are open to public and free of cost.

But apart from that, in order to have more generic interpretation, these system could have used variety of bug repositories from various companies.

We have observed one more fact here, as this field might include cognitive habits of each individual or working environment features while reporting bugs, no paper has considered this fact. One reason behind this could be the fact that this is unlikely that company would maintain information like individual habits or environmental features for bug report filing task.

BASELINE RESULTS

Not all papers we studied had baseline results to compare against. For example, for paper 4 in which authors had novel idea of considering duplicate as complimentary information providers where there were no study before, hence no baseline results were available. Similarly, for paper 7 in which authors claimed to automatically assign bugs to corresponding developer, they did not find any baseline results.

Results of paper 9[4] are compared with results of first paper [17]. Author claims that contextual approach improves accuracy in detecting duplicate bugs up to 11.55%. Reason for selecting firsts paper [17] for comparing results is first paper uses textual similarity very effectively for duplicate detection. Author have wanted to compare results with model which makes use of textual similarity of bug reports very efficiently.

STUDY INSTRUMENTS

The papers do not use any study instruments. Our recommendation is to use interviews, surveys before and/or after bug report submission in addition to the work done above or as an additional study on top. This would help us to gain cognitive information and information about the environmental behavioral data and contextual data about the end user who is submitting bug reports. This could help us figure out why and under what conditions a user submits duplicate bug reports and why the report may contain incorrect data. If we studied this, we could modify the environmental or psychological factors which lead to such behavior. Interviews can include a small set of questions asking very basic information like - did you read basic guidelines before filing the report? - Were informative suggestions provided by system useful? Surveys might get us any relevant data considering while reporting bugs. Also, this could help us identify what the users consider irrelevant while filing a report. These studies could be based to make recommendations on how to improve user acceptance of bug tracking systems and hence improve the acceptability of these bug tracking systems which will in turn improve the manner they are used in.

Almost all paper we studied have used bug repositories alone and have not considered using any such study instruments listed above.

We observed that most of the papers have used most popular term frequency models like BM25F and varied version in IDF (Inverse Document Frequency) techniques. Experimental data can be however be also tested against the study instruments listed above and these results can be used as baseline results to state improvements/ efficiency of newly suggested approach/ model.

OUR COMMENTARY

This section has our interpretations/ modifications /observations about all the papers we studied in this course.

We interpreted that the best bug tracking system to retrieve duplicate reports to consist of these three steps: Cleaning the data, training a model to identify duplicates and retrieving duplicate reports. Bugs reported are passed to one or the other model to find out any redundancy.

In paper 3 [19], though the authors claimed that execution information is equally important as that of natural language information about bugs, their study methods are leaning towards in favor of natural language processing. In assigning classification based heuristics, the highest ranked class is the one which favors NLP similarity compared to Execution Information similarity. This contradicts their initial stand saying both the categories: execution information and natural language information should get equal importance. In some of bugs, neither natural language information nor execution information may be the reason for a bug report. This is a serious consideration the authors did not handle. A bug may also be reported because a user has am interaction with the user interface which has a bad design and behaves unexpectedly and gets reported as a bug .In this case, relevant information of user's interaction with system should have been taken into account.

In paper 5 [7], authors have claimed that using 54 textual features might lead them towards more accurate determination of duplicate bugs. There is no comprehensive reason given as to why only 54 features were used or why only these particular 54 features. There may be irrelevant data created if one considers all 54 features for determination. This adds up in adding unnecessarily introduced noisy data which would need to be cleaned out later. Including relevant data should have sufficed here.

In same paper [7], the suggested approach considers all new reported bugs as duplicates of master report and further work is done accordingly. We think that more relevant bug which is having as much as possible relevant data should be considered as master report and all other newly reported bug should be linked to that master report as sub-reports this makes sense as the master report in bucket of duplicates represents all bugs in a list. Master report should have most of the relevant data about a particular bug. Other slave duplicates just acts as complimentary information items as suggested by Betterberg et al[3].This reduces time for finding duplicates as newly reported bug will be compared to most relevant master report initially and removes need to scan entire bucket of duplicate bugs. Only one comparison would be needed in such a case. Hence, we recommend building the work of [7] based on work by [3]

In paper 7 [20], an automated approach was suggested to assign appropriate developer to bug for bug fixing. There might be serious issue where certain amount of bugs are assigned to same developer irrespective of number of bugs he is dealing at the same moment. This phenomenon is called as 'Biased Assignments' where a particular developer is constantly assigned to bugs even though he is not finished with old bug issues. Authors have not considered this scenario while designing DevRec (the new tool proposed in the paper). This will lead to incorrect resource allocation again which defeats the purpose of the study altogether. We propose having some kind of scheduling mechanism on top of the approach suggested.

In same paper [20], bug resolver is new term coined which states a bug resolver can be anyone who participate/ or contributes in bug resolving. Authors have omitted differentiation between these types of bug resolvers. This is a very important classification which should have been taken into account by the authors. Participants might not help in getting insight details of bugs for specific developer assignments as the developer .Hence, both cannot be put under the same bucket or category.

Also, the authors have missed the fact that only those people who have been working on a domain for a considerable time be considered when assigning a bug. A relatively new developers may fix a bug blindly by local correction without thinking of global impact of that change. This could add errors in addition when resolving a single error. We recommend spending resources on introducing a biased assignment to save resources in the long run and avoid introduction of new errors.

In paper 3[19], this experiment has not taken into account a variety of bug reports for analyzing bugs. Authors have claimed execution information plays vital role in determining duplicates. However, each company has different standards, execution environments. Application specific workflows. Authors have not taken these differences into consideration. These aspects should be considered when building a robust bug duplicate detection system.

Also, we recommend doing surveys to take into account user behavior and using that data to conduct analysis into why a bug is reported twice and why a bug is reported incorrectly.

PATTERNS / NEGATIVE PATTERNS / NEGATIVE RESULTS

In this section, we list out the various results or anti results the authors enlisted and the reasons for those and in some cases the methods to resolve those.

In paper 4 [3], while claiming the theory of merging duplicates, authors have stated some of the negative results as well. Authors are afraid of Results that may not generalize other projects. Sometime addition information slows down the process. It may be harmful and disturbing.

Authors [3], claimed two types of threats. Threats to external validity concern their ability to generalize from this work to general software development practice. These threats includes

- Non-generalization to other projects
- Noisy data which might be harmful for analysis.

Other type is internal threats which concern the appropriateness of our measurements, our methodology and their ability to draw conclusions from them. These threats includes

- Correctness of data,
- Obsolete attachments,
- Implicit assumptions on the triaging process,
- Validation setup,
- Chronological order of duplicates,
- Biased results
- Resubmission of identical bug reports.

Biased assignment of bugs[20] to some of the developer might be negative result from authors approach.

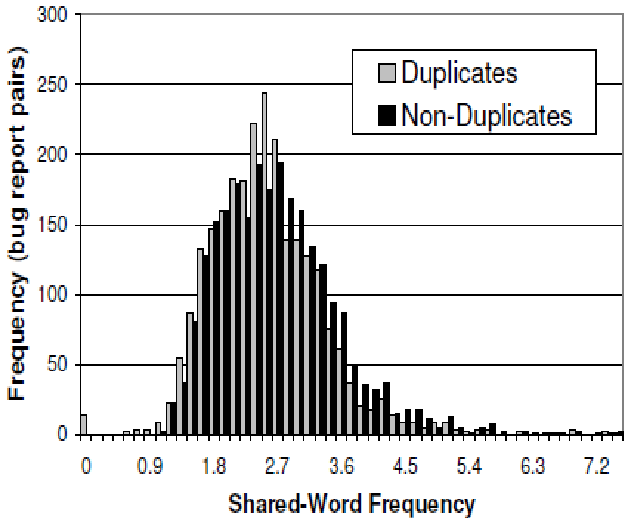
In paper 1[17], authors had found two patterns for handling triaging procedure.

- Filter duplicates before reaching triagers: This approach reduces traigers overload and if accurately filtering algorithm is used, it increases efficiency.
- When a new bug reported, Provide top-k similar bugs for investigation: This approach supports Bettenburg et al's thought that one bug report might only provide partial view of defect, while multiple bug reports complements each other.

Authors of paper[17] had gone with approach where top-k retrieval of similar bug report would accurately report duplication.

In paper 3[19], the authors have mentioned only 3 techniques for collecting execution information. Bugs that are not covered under none of those techniques will never be analyzed by author's novel idea. So without generalization, this approach can bring negative results in determining duplicates.

In paper 2[18], authors have performed analysis of shared word frequency between duplicate-original pairs (light) and close duplicate-unrelated pairs (dark). The main reason behind this analysis to decide why the inclusion of inverse document frequency was not helpful in identifying duplicate bug reports. The shared-word frequency between two documents is the sum of the inverse document frequencies for all words that the two documents have in common, divided by the number of words shared.



The above figure, picked from [18] shows the two distributions. The distribution of duplicate-unique pair values falls to the right of distribution of duplicate original pair. On conclusion, shared frequency is more likely to increase the similarity between unrelated pairs than of similarity between duplicate original pairs. Which in turns leads authors to dismiss shared-word frequency from consideration as a useful factor to distinguish duplicates from non-duplicates

Each company has their own cultural environment, coding standards. On addition to that each individual has own habits while dealing with code. These patterns plays an important role in determining duplicates. This patterns should be given consideration while deciding duplication of bugs. If a new bug is reported, user specific workflows can be applied to corresponding bugs in order to find if bug repository has duplicate bug or not. This might expedite things little bit at first place.

NEW RESULTS

In this section we will focus on obtained results which has potential future problems.

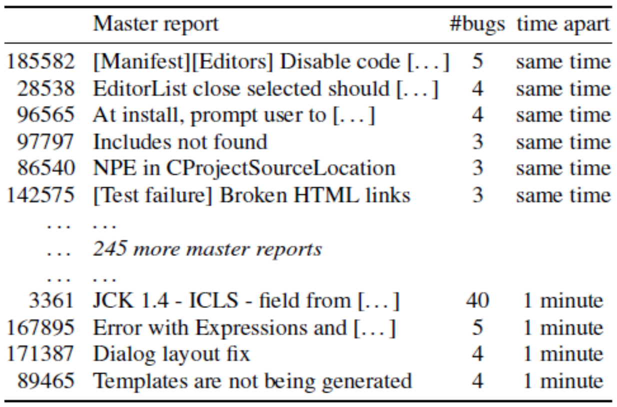
In paper 1[17], author have predefined fixed set of fields from bug reports for determining potential duplicates. But in future, fields in bug report can be changed. There is no doubt that these bug filing methods would change in nearby future as these methods takes data from users without any preprocessing and without any intelligent. There should be provision of dynamic addition of extra field to collect more data from user.

In various approaches where top-k similar bug reports were retrieved, there should be a meaningful way to determine master report from those top-k retrieved bug reports.

An attempt to find out root cause of a bug should be made after finding top-k related bug retrieval. As at the point after analysis and finding top-k related bugs, one has probably more than enough relevant data about a particular bug. This crucial data with the help of intelligent algorithm having domain contextual data might lead us the root cause of bug.

No paper in this domain consider Human Computer Interaction factor while determining duplicates. As it might involves totally new feature to consider in analysis. Very obvious features would be users various perceptive abilities, distinct cognitive capabilities should be considered before deciding a reported bug a duplicate or not.

In case of duplicates which are reported at the same time, cannot be compared against potential candidate and hence result in considered as non-duplicate bugs.



	Master report	#bugs	time apart
185582	[Manifest][Editors] Disable code [. . .]	5	same time
28538	EditorList close selected should [. . .]	4	same time
96565	At install, prompt user to [. . .]	4	same time
97797	Includes not found	3	same time
86540	NPE in CProjectSourceLocation	3	same time
142575	[Test failure] Broken HTML links	3	same time
. . .	. . .		
. . .	*245 more master reports*		
. . .	. . .		
3361	JCK 1.4 - ICLS - field from [. . .]	40	1 minute
167895	Error with Expressions and [. . .]	5	1 minute
171387	Dialog layout fix	4	1 minute
89465	Templates are not being generated	4	1 minute

This (above figure) is an example from paper [3] showing column "time apart" lists the time that the first and last bug report are apart in the corresponding duplicate groups. The table is sorted in ascending order by "time apart" and show that many duplicates are reported at exactly the same time as the original master report.

CONCLUSION

Maintaining huge bug repositories is very tedious job in large software companies. Due to duplicate bug reports, there is unnecessary wastage of human resources as more than developers will be assigned to same bug report and resources would be spend on resolving same bug. Traditionally, human triagers used to detect duplicate bugs by manual way and if bug is determined as duplicate then it is discarded otherwise it is assigned to corresponding developer. Considering large amount of bug reported daily, manual triaging would take significant amount of time. Hence, there is need to automate triaging process.

To determine duplicates, bug reports can be compared to each other, using various approaches suggested in all papers throughout this course. Each approach has its own strength and weakness, there will be always tradeoff in finding duplicates and accuracy in results. Till date, manual triaging has outperformed the automation procedure in term of accuracy in finding duplicates.

FUTURE SCOPE / OUR RECOMMENDATION

Bug reports are nothing but textual representation of issues found in system. Considering large number of bugs filed every day and variety of bug reporting types, this adds up very huge textual data. In order to process such humongous data, initial preliminary procedures should be always executed. These preliminary procedures are stemming, removal of stopping word, removing noisy data etc. This saves lot of time and improves overall efficiency of duplicate detection system.

For simplicity or just for the initial runs, most of authors have simply neglected invalid bugs. Reason behind invalid bugs might be non-reproducible, unable to follow test case, workflow changed etc. Author have straight forwardly discarded those invalid bugs while detecting duplicate bugs. Sometimes due to some minor mistakes while reporting a bug, bug can be considered as invalid bug. A little attention to those invalid bugs should be given in order to have a fully functioning bug detection scheme.

If bug reporting habits were considered, it would be easy not only to improve accuracy in determining duplicate bugs but also it helps to do assignment of bugs to corresponding developer.

Variety of data should be analyzed and then analyzed data should be used for tuning model which decides whether newly reported bug is duplicate or not. These data sets might include contextual information, domain information, module based system vocabulary, user's cognitive habits, cultural constrains specific to individual companies, environmental behavioral and bug filling practices.

More importantly, in order to have more generic atomizer for determining duplicates, a variety of bug repositories should be tested against. All the papers we studies in this course for this area, were using mostly open source bug repositories from either of Mozilla, Eclipse, OpenOffice or open sourced android projects.

Some of the papers recommends top-k bug retrieval and maintain master-slave bugs for that bucket of bug reports.

Master slaves should be selected in such a way that it should represent more relevant information about a bug that any other bug report in that particular bucket. All authors had taken first bug report as master bug report and every single newly reported bug report as slave bug reports. There might be highly chance that newly reported bug report might have more relevant data than master report.

Only few paper have studied time complexities in finding duplicates. There were no proper explanations found behind omission of calculating time complexities. As it is pretty obvious, considering huge number of bugs reported every day, these system should be quick enough to take decision on each bug report. Same is the observation with space complexities. De-duplication systems should be capable of working in fair amount of memory space to decide whether a reported bug is duplicate or not.

We recommend that contextual information should be taken into consideration while deciding duplicates. Duplicate bug has similar contextual information even though theirs textual representation shoes vivid differences.

REFERENCES

[1] T. Nakashima, M. Oyama, H. Hisada, and N. Ishii, "Analysis of software bug causes and its prevention," Information and Software Technology, vol. 41, no. 15, pp. 1059–1068, 1999.

[2] S. Hangal and M. Lam, "Tracking down software bugs using automatic anomaly detection," in Proceedings of the 24th international conference on Software engineering. ACM, 2002, pp. 291–301.

[3] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicatebug reports considered harmful really?" in Software Maintenance, 2008. ICSM 2008. IEEE International Conference on. IEEE, 2008, pp. 337–345.

[4] A. Alipour, A. Hindle, and E. Stroulia. "A contextual approach towards more accurate duplicate bug report detection". Proc. of the Tenth Intl Workshop on Mining Software Repositories, pages 183–192, 2013.

[5] 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.

[6] 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.*

[7] 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.

[8] A. Sureka and P. Jalote, "Detecting duplicate bug report using character n-gram-based features," in Proceedings of the 2010 Asia Pacific Software Engineering Conference, 2010, pp. 366–374.

[9] J. Sutherland. "Business objects in corporate information systems." *ACM Comput. Surv., 27(2):274–276, 1995.*

[10] P. Runeson, M. Alexandersson, and O. Nyholm. "Detection of duplicate defect reports using natural language processing". *In International Conference on Software Engineering (ICSE), pages 499–510, 2007.*

[11] A. Hindle, N. Ernst, M. W. Godfrey, R. C. Holt, and J. Mylopoulos, "Whats in a name? on the automated topic naming of software maintenance activities," *submission: http://software process.es/whats-in-aname,vol. 125, pp. 150–155, 2010.*

[12] D. Blei, A. Ng, and M. Jordan, "Latent dirichlet allocation," *the Journal of machine Learning research, vol. 3, pp. 993–1022, 2003.*

[13] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic, "An information retrieval approach to concept location in source code," *in Reverse Engineering, 2004. Proceedings. 11th Working Conference on. IEEE, 2004, pp. 214–223.*

[14] P. Hooimeijer and W. Weimer. "Modeling bug report quality". *In Automated software engineering, pages 34–43, 2007.*

[15] P. Runeson, M. Alexandersson, and O. Nyholm. "Detection of duplicate defect reports using natural language processing". *In International Conference on Software Engineering (ICSE), pages 499–510, 2007*

[16] N. Mishra, R. Schreiber, I. Stanton, and R. E. Tarjan. "Clustering social networks". *In Workshop on Algorithms and Models for the Web-Graph (WAW2007), pages 56–67, 2007.*

[17] C. Sun, D. Lo, S. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," *in Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering. IEEE Computer Society, 2011, pp. 253–262.*

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

[19] 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 '08: Proceedings of the 30th International Conference on Software Engineering. ACM, 2008.*

[20] X. Xiaz, D. Loy, X. Wang, and Bo Zhoux"Accurate developer recommendation for bug resolution," in *WCRE, 2013, pp. 72–81.*

[21] A. Nguyen, D. Lo, C. Sun, "Duplicate Bug Report Detection with a Combination of Information Retrieval and Topic Modeling" in Proceeding MSR '13 Proceedings of the 10th Working Conference on Mining Software Repositories Pages 183-192