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* – A very large software systems is always evolving consistently throughout the lifetime of system. This leads to continuous reporting of bugs and apparently resolving of bug. Sometime same issue can be reported more than once. There can be two reports having almost same issues but claiming to be different. There are multiple reasons for duplicate bugs to exist. Maintaining such kind of bug repository is painful from maintenance and supervision perspective. There has been number of studies on automation of this job where triaging can be done for detecting duplicate bugs. There has been studies on different areas like automatic bug assignment to specific developer, use duplicate bug reports to generate more relevant data for resolving bugs, use duplicates to resolve unfound view of bugs.**

Here, we will skims through all papers we have studied in this course to explore work done in this domain. All the papers have work/research in duplicate bug detection area. We will also comment on 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

Considering today's technology evolution, computer technology is found in almost every aspect of day to day life. Sometime due to complexity or range of tasks performed, these system becomes humongous. Maintaining such large system is one difficult task to do. In addition to that, as time goes, computer systems evolves. Events responsible for those constant evolution might be detection of bugs, enhancements of current features and addition of new requirements. Here we will concentrate on detection of bugs and its impact on system maintenance.

Bugs are deficiency of code written to fulfill required, expected and meaningful behavior of system. Bugs can be defined as incapability of code in delivering promised functionality. Bugs occurs for many reasons. Some of the obvious reasons could be bad design of software, careless developing practices, incomplete requirement analysis of system and/or incomplete/improper testing etc[1],[2]. Every company has their own tool for reporting bugs. There are commercial tools as well for maintaining bugs like Bugzilla. Bugzilla is a web-based general-purpose bug tracker and testing tool originally developed and used by the Mozilla project, and 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 added into bug repositories periodically. Most of bugs are reported in testing phase. Reported bugs are analyzed and assigned to appropriate developers/resources for corrections. After fixing bugs, reported bug is made inactive.

There is likely a chance that same bug is reported by more than once. Several reasons such as such as lack of motivation of users and defects in the search engine of the bug-tracking systems [3], duplicate bugs occurs in bug repository. More than one bugs can report same bug behavior but in different ways. Same bug can be reported in different contexts or in different environments. In case of undetected duplicate bugs, there might be chance that more than one developers are assigned to address same issue. These assignment leads to unnecessary work towards solving same issue, leading to wastage of resources.

When a bug is reported in bug repository, it is scanned through triager. Triager is person who decides reported bug is duplicate or not. Each reported bug checked against all present bugs in repository. If reported bug is duplicate, then it is marked as duplicate and handled accordingly. Otherwise, it is added in bug repository and assigned to appropriate developer after bug analysis. Manually triaging would take considerable amount of time and results are unlikely to be accurate. If we consider an example of Mozilla in 2005, it was reported that, "everyday almost 300 bugs appear that need triaging. This is far too much for only the Mozilla programmers to handle" [5]. This might lead to inefficient use of resources. Considering large number of bugs reported daily and amount of work at triager's end, there has been number of studies carried out in order to automate process of tedious duplicate bug report detection [4]. We will talk about how a bug is analyzed and assigned to appropriate developer. There is possibility to automate this process as well where we have enough data about a bug so that corresponding developer can be found in advance.

Though duplicate bugs do carry mostly redundant data, they are not necessarily bad. But there might be some findings from duplicates which can help us to view different aspects of bugs. Duplicate bugs can give us more insight details about a functionality that is not working. 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 are compared against each other and concluded about duplicity. Most common form for this comparison is to consider their textual based representations. Sometimes ranking among bug reports is calculated in order to state extent up to which given reports are similar to each other. Studies in [6], [7] and [8] processes only 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-k related bugs can be found for each new bug reported and if reported bug is marked as duplicate then it can be assigned to that bucket [6]-[8].

MOTIVATION

As software systems grows large, it becomes more difficult to maintain code repositories. In addition of duplicate bugs, it just adds more maintenance work. Maintenance activities account for over two thirds of the life cycle cost of a software system, summing 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 amount on maintaining bug repositories is not considered as wise distribution. In case of duplicates, more than developer can work on fixing same issue. Same is the case with testers, who are testing same issue at the same time, leads to unnecessary wastage of resources. In addition to that, there were no robust techniques for comparing two bug reports in order check duplicates. Most obvious technique would be comparing textual representation of both bug reports [10]. This might include finding similarities between 2 reports using similar words, fields used in reports, technical words specific to domains. But again, these features cannot state duplicity of reports with accuracy. It involves processes like removal of stopping words, get rid of noisy data in report, stemming as initial clean up. Overall accuracy in finding duplicates not only depends on algorithm used for finding textual similarity, but also how effectively initial clean-up is done.

This motivates 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.

HYPOTHESIS

In present studies, we hypothesize that finding an accurate and stable approach for finding duplicates and automating them would improve overall project lifecycle management system. We also predict, by assuming appropriate accuracy in algorithm, most bugs can be assigned automatically to respective developers to work further. There is large chance of discovering hidden and/or obscure details of bugs from duplicate bugs, as sometimes duplicate bugs compliments each other. At the end, if one gets enough insights for a particular bug using most appropriate approach, an attempt can be carried out to determine root cause of bug

RELATED WORK

Information Retrieval is 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 IR and bug deduplication.

A. *IR in Software Engineering*

Hindle et al. have 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), 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 it is a fast algorithm with reasonable performance.

Marcus et al. have 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 has been issues with interface of bug tracking software itself, which causes less precision while reporting bugs [13]. There might be some of the fields which are not available while filing a bug report. In this case, user might get confused and leads in unpredictable data inputs. In order to submit all of the observations in restricted fields might cause addition of irrelevant data in certain fields. Due to unavailability of fields while reporting bug reports,

In first paper [17], Sun et al. has proposed a new retrieval function (REP) for finding textual similarity between two bug reports. For more accurate measurement of textual similarity, they have extended BM25F – an effective similarity formula in information retrieval community. 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. These supplementary fields adds more relevance in finding similarity. On 209058 reports from Eclipse, authors claims that they achieved recall rate @ of 37-71% and mean average precision rate of 47%. They have followed an approach where when new bug is reported, top-k similar bug reports were retrieved which supports Betternburg's et al.'s thought that multiple bug reports complements each other.

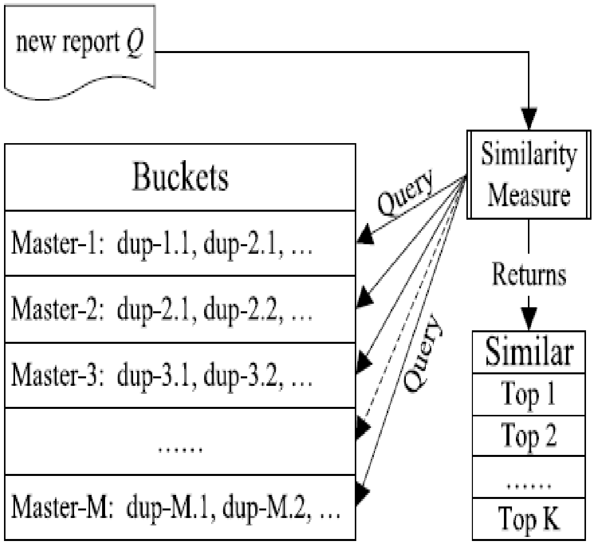

new report Q
Buckets
Master-1: dup-1.1, dup-2.1, ...
Master-2: dup-2.1, dup-2.2, ...
Master-3: dup-3.1, dup-3.2, ...
......
Master-M: dup-M.1, dup-M.2, ...
Query
Query
Similarity Measure
Returns
Similar
Top 1
Top 2
......
Top K

Fig. 1. Overall Workflow for Retrieving Duplicates

In above figure, authors have queried new report in bug repository in order to retrieve top-k similar reports which complements each other. Using similarity measures, each bug reported is mapped for further queries. 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 our second paper [18], Jalbert et al. have proposed an automation system which uses surface features [14], textual similarity metrics [15] and graph clustering algorithm [16] combined to identify duplicate bugs. Author 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 for each 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.

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

Surprisingly authors found that employing IDF in finding duplicates, resulted in strictly worse performance than a baseline using only non-contextual term frequency.

In our opinion, word frequency would not be the sole reason in comparision of duplicates. Consider an example of two totally different bug report from same domain. In this case, as these two bug reports are from same doamin, 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 varition found but in terms of numbers. But there will not be any significant differences in term frequencies as for explaining certain tasks from same doamin, certain words should be repeated to describe the action.

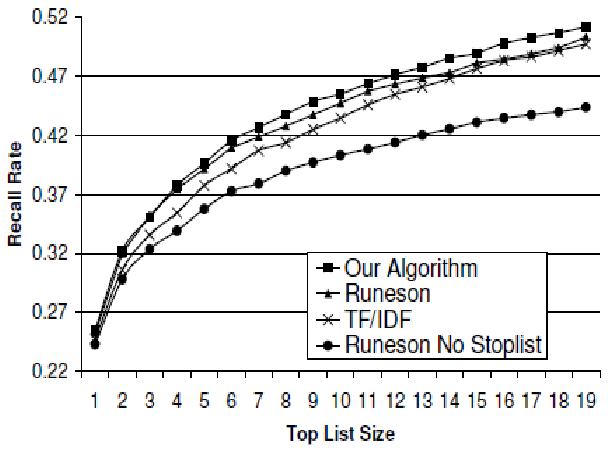


Figure 2. Recall rate of various algorithms on the Runeson task as a function of top list size. Our algorithm performs up to 1% better than that of Runeson et al., which in turn performs better than a direct application of inverse document frequency (TF/IDF).

Above figure strictly shows the recall rates obtained as a function of the top list size. The TF/IDF algorithm uses the TF/IDF weighting that takes inverse document frequency into account; it is a popular default choice and serves as a baseline. Authors have considered the Runeson et al. algorithm with and without stoplist word filtering for common words. Unsurprisingly, stoplists improved performance. It is interesting to note that while the TF/IDF algorithm is strictly worse at this task than either algorithm using stoplisting, it is better 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 dismieed this approch and headed towards weightage based approach where each appearing term in report is given weigh and updated on timely maneer.

In out third paper [19], Wang et al. have gone one step ahead and included excution information as well while determining duplicates. Authors found that only one of them (Natural Language processing and Execution Information) cannot be used for finding similarity efficiently among bugs. They had come up with some heuristics which ranks/weights the potential duplicates with combination of both approach. 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 our 4'th paper[3] Betternburg et al. had come up with very interesting thought claiming that, each duplicate is not necessarily just a redundant piece of information. These duplicates can compliments each other and provide very crucial information about getting more insight for a specific bug. Authors have proposed an approch where one can maintain master report and extended master reports which serves as information items in retrieving more relevant infotmation. Authors had identified some of the potential reasons behind bug reporting as below.

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

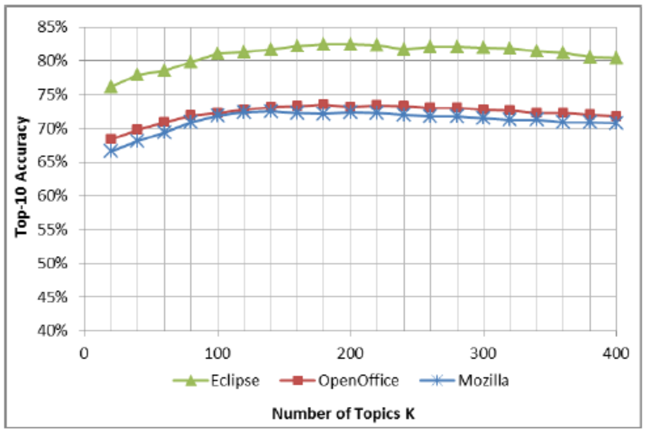
Some of the obsercations as below:

- 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 our 5'th paper[7], Sun et al. have concentrated on various similarity features which are distinct properties of a particular bug report. They have reached upto uch 54 distinct similarity features and used discriminative model to automatically assign weights to each feature to discriminate duplicate reports from nonduplicate ones, which is rigorous and has theoretical support from machine learning area. Author claims that thres approch is more adaptive, robust and automated as 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 the 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 similarity features, authors have used Fisher's score having idf-based fomulas. 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. Which has leaded them towards selection of idf based formnulas as feature scores.

In out 7'th paper[20], Xia et al. have one more step ahead and introduce a noval idea of automatically assignment of duplicate bugs to corresponding developer. They have 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@5 and recall@10 scores of Bugzie (old existing too used by author as base comparison) by 57.55% and 39.39%, respectively. DevRec also outperforms DREX (old existing tool used by author as base comparison) by improving the average recall@5 and recall@10 scores by 165.38% and 89.36%, respectively.

In our 8'th paper[21], Nguyen et al. introduced DBTM, a duplicate bug report detection approach that takes advantage of both IR-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). 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.



Above figure shows work of authors on evaluation of the sensitivity of DBTM's accuracy with respect to different numbers of topics K. They ran DBTM on 3 different bug repositories from anemly 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 other 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 a 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 deermine 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 globallya and locally.

In our 9'th paper[4], Alipour et al. has proposed an approach to consider contextual information as well while determining duplicates. They have extended the state of the art by investigating how contextual information, relying on our 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-deduplication 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. Author have paid enough attention in initial clean up of overall information. As a part of intial configuration, each report is preprocessed to remove stop words and stemmed. 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 firsts paper 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.

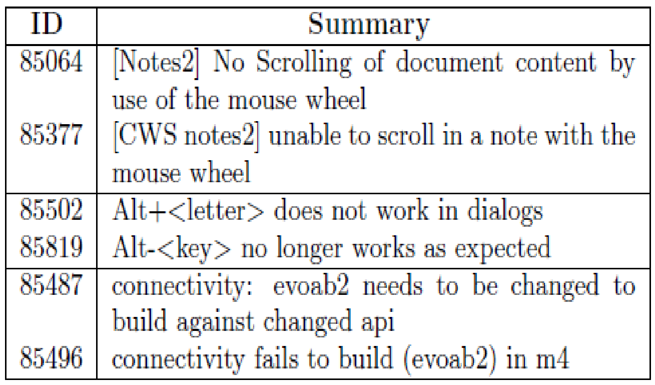
CHECKLISTS

Checklists are minimal datasets and corresponding actions considered to 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) is very typical example of how duplicate bugs looks like. As we can see, thought textual reprensation of these bug looks different, they convery same message. If we scan above bug reports from contextual perspective, it can be stated vividly 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 ataht, 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 paper we studied in this domain, has bug reports looks like above having textual representation. Bug repositories having such bug reports distinguished by their unique id’s.

Surprisingly 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

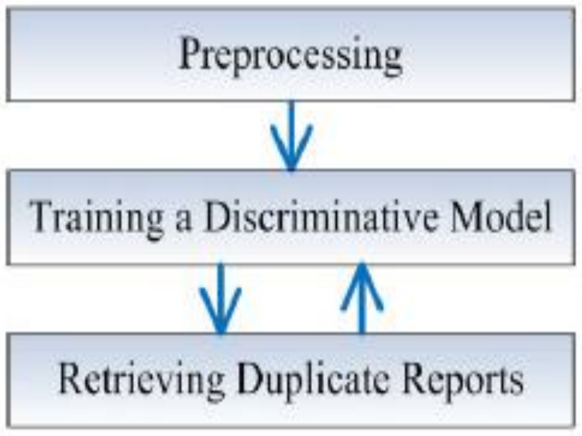
We strongly recommends to use interviews, surveys before and/or after bug report submission. This would help us to gain cognitive information and environmental behavioral data about end user who is submitting bug reports. Interviews might include a small set of questionnaire asking very basic information like did you read basic guidelines wile filing report? Were informative suggestions provided by system useful? etc. Surveys might get us any irrelevant data considering while reporting bugs.

Almost all paper we studied have used bug repositories blindly while experimentation without considering above facts.

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 tested against these module and 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.



This is (above figure) how we interpret overall framework to retrieve duplicate reports. This is very abstract top level view of any generic duplicate retrieval system. Each reported bug is then passed to discriminative model in order to find out any redundancy.

In paper 3 [19], Yet author’s claim says execution information is equally important as that of natural language information about bugs, their heuristic methods are tending towards in more favor of natural language processing. In classification based heuristics, highest ranked class is the one, which favors NLP similarity compared to Execution Information similarity. Which contradicts their initial stand saying equal importance. In some of bugs, neither natural language information nor execution information is always superior to the other in all cases. This is very serious issue that authors had completely forgotten to handle while doing analysis of new bug reports. A common example could be a bad user interface design which might leads to unexpected behavior and apparently reported as bug. In this case, relevant information of user’s interaction with system

In paper 5 [7], authors have claimed that using 54 textual features might lead them towards more accurate determination of duplicate bugs. But there is no comprehensive analysis of fact that why did they choose those many features? It has high chance of creating irrelevant data if one considers all 54 features for determination. This adds up in handling unnecessarily introduced noisy data, which could not have been included at first place. For some bug reports, only relevant field suffice to get enough data for analysis.

In same paper [7], 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 repost as slave duplicate report. This makes sense as master report in bucket of duplicates represents all bugs in that 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.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.

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 (name of theirs new tool).

In same paper [20], bug resolver is new term coined which state a bug resolver can be anyone who participate/ or contributes in bug resolving. Authors have omitted differentiation between these types of bug resolvers. Participants might not help in getting insight details of bugs for specific developer assignments. Authors have missed the fact that, sometimes developers fix the bug blindly by local correction without thinking of global impact of that change. These developer should not be considered as bug resolver. This happened a lot with new developers which are relevantly new to domain in which issue has been notified. We recommend to categorize only those people who has been working with domain in which issue has been notified for significant amount of time. Spending resources on resolving biased assignment is considered as compromising saved resources in first place.

In paper 3[19], this experiment has lack of variety of bug's reports for analyzing bugs. As authors have claimed execution information plays vital role in determining duplicates, each company has different standards/execution environments/application specific workflows. Authors did not mention anything about that. It would be nice to consider those aspects as well in order to have robust bug duplicate detection system.

PATTERNS / NEGATIVE PATTERNS / NEGATIVE RESULTS

In paper 4 [3], while claiming this 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.

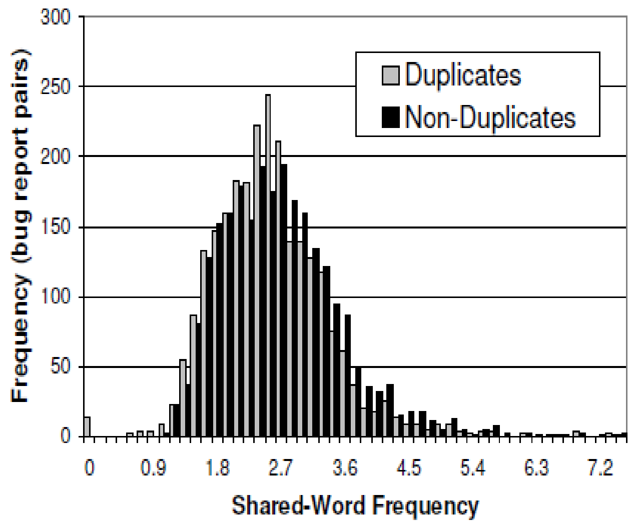
In paper 1[17], authors had found two patterns for handling triaging procedure.

- 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.



Above figure 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.

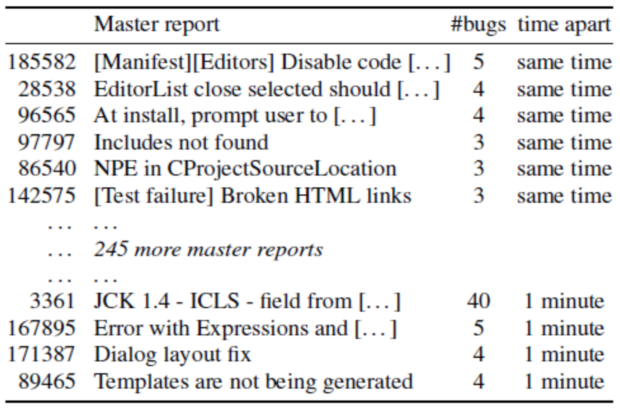
In paper 1[17], author have predefined fixed set of fields from bug reports for determining potential duplicates. But in future, fields in bug report can be changed. There is no doubt that these bug filing methods would change in nearby future as these methods takes data from users without any preprocessing 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