Research Journal of Applied Sciences 13 (9): 522-531, 2018
ISSN: 1815-932X
© Medwell Journals, 2018

Detecting Duplicate Bug Reports Techniques

[1]Zahra Aminoroaya, [2]Behzad Soleimani Neysiani and [3]Mohammad Hossein Nadimi Shahraki
[1]Department of Software Engineering,
Allame Naeini Higher Education Institute, Naein, Isfahan, Iran
[2]Department of Software Engineering, Faculty of Electrical and Computer Engineering,
University of Kashan, Kashan, Isfahan, Iran
[3]Faculty of Computer Engineering, Najafabad Branch, Islamic Azad University,
Najafabad, Isfahan, Iran

Abstract: With the advent of the Internet and the spread of computer users, many applications have been developed that are used by millions of user's everyday tasks like office applications or web browsers. Software companies spend over 45% of cost in dealing with software bugs. An inevitable step of fixing bugs is bug triage which aims to correctly assign a developer to a new bug. Bug-tracking and issue-tracking systems tend to be populated with bugs, issues or tickets written by a wide variety of bug reporters with different levels of training and knowledge about the system being discussed. Many bug reporters lack the skills, vocabulary, knowledge or time to efficiently search the issue tracker for similar issues. As a result, issue trackers are often full of duplicate issues and bugs and bug triaging is time consuming and error prone. Software bugs occur for a wide range of reasons. Bug reports can be generated automatically or drafted by user of software. Bug reports can also go with other malfunctions of the software, mostly for the beta or unsteady versions of the software. Most often, these bug reports are improved with user contributed accounts experiences as to know what in fact faced by him/her. Addressing these bug's for the majority of effort spent in the maintenance phase of a software project life cycle. Most often, several bug reports, sent by different users, match up to the same defect. Nevertheless, every bug report is to be analyzed separately and carefully for the possibility of a potential bug. The person responsible for processing the newly reported bugs, checking for duplicates and passing them to suitable developers to get fixed is called a Triager and this process is called Triaging. The utility of bug tracking systems is hindered by a large number of duplicate bug reports. In many open source software projects as many as one third of all reports are duplicates. This identification of duplicacy in bug reports is time-taking and adds to the already high cost of software maintenance. To decrease the time cost in manual work, text classification techniques are applied to conduct automatic bug triage. This study presents an overview of the works done to better detect duplicate bugs have been conducted on open source data set.

Key words: Bug report analysis, duplicate bug report detection, feature extraction, information retrieval, text mining, domain knowledge, execution information, classification

INTRODUCTION

Due to complexities of software systems, software bugs are prevalent. To manage and keep track of bugs and their associated fixes, bug tracking system like Bugzilla1 has been proposed and is widely adopted. With such a system, end users and testers could report bugs that they encounter. Developers could triage, track and comment on the various bugs that are reported. A bug tracking system or defect tracking system is a software application that keeps record of reported software bugs in software development projects. It may be considered as a type of issue tracking system. Most of the bug tracking systems, such as those used by many open source software projects, allows end-users to enter bug reports directly (Zimmermann *et al.*, 2009). Other systems are used only inside a company or organization involved in doing software development. In general bug tracking systems are integrated with other software project management applications.

The database of reports is critical for many software projects as it describes both bugs that need to be fixed

Corresponding Author: Zahra Aminoroaya, Department of Software Engineering, Allame Naeini Higher Education Institute Naein, Isfahan, Iran

and the new features to be added. Open source software projects typically have a bug repository that allows both developers and users to post problems encountered with the software, suggesting of possible enhancements and commenting upon existing bug reports. Many popular open source projects get vast number of bug reports (Sun *et al*., 2011).

As new software systems are getting larger and more complex every day, software bugs are an inevitable phenomenon. Software development is an evolutionary process where after the first release, bug report submissions by the users and testers come through. Bugs arise during different phases of software development, from inception to transition. They occur for a variety of reasons, ranging from ill-defined specifications to carelessness to a programmers misunderstanding of the problem, technical issues, non-functional qualities, corner cases, etc. Also, software bugs are considerably expensive. Existing research indicates that software bugs cost United States, billions of dollars per year (Sun *et al*., 2011).

Defect (also referred to as issue or bug) reporting is an integral part of a software development, testing and maintenance process. Typically, bugs are reported to an issue tracking system which is analyzed by a Triager (who has the knowledge of the system, project and developers) for performing activities like; Quality check to ensure if the report contains all the useful and required information, duplicate detection, routing it to the appropriate expert for correction and editing various project-specific metadata and properties associated with the report (such as current status, assigned developer, severity level and expected time to closure). It has been observed that often a bug report submitted by a tester is a duplicate (two bug reports are said to be duplicates if they describe the same issue or problem and thereby have the same solution to fix the issue) of an existing bug report. Studies show that the percentage of duplicate bug reports can be up-to 25-30%. This can significantly hamper the bug fixing process and product release.

Recognizing bugs as a "Fact of life", many software projects provide methods for users to report bugs and to store these bug/issue reports in a bug-tracker (or issue-tracking) system. The issue-tracking systems like Bugzilla and Google's issue-tracker enable the users and testers to report their findings in a unified environment. These systems enable the reporters to specify a set of features for the bug reports such as the type of the bug report (defect or feature request), the component in the system the report belongs to, the product the report is about, etc. Then, the developers will select the reported bugs considering some of their features. The selected bug reports are handled with respect to their priority and eventually closed. The issue-tracking systems also provide users with the facility of tracking the status of bug reports. Addressing bug reports frequently accounts for the majority of effort spent in the maintenance phase of a software project's life-cycle. This is why, researchers have been trying to enhance the bug-tracking systems to facilitate the bug-fixing process.

For most of the complex software, more bugs are reported than can be easily handled. Each report needs to be triaged by a person, known as triager (Anvik *et al*., 2006) to determine if reports are meaningful and if it does, it must be assigned to an suitable developer for further handling. Moreover, one of the most important task of triager is to identify bug reports if these are duplicates of some previously submitted report related to some still-uncovered bug. In large projects where there are thousands of reports to search through, identifying duplicate bugs can be an expensive task. The use of a bug repository can improve the development process in a number of ways:

- It allows the evolution of the project to be tracked by knowing how many reports are outstanding
- It allows developers who are geographically distributed to communicate about project development
- It enables approaches to determine which developers have expertise in different areas of the product
- It can help improve the quality of the software produced and
- It can provide visibility to users about the status of problem reports

The bug repository can thus provide a location for users, quality assurance teams, developers and managers to engage in a user-integrated development process. However, the use of a bug repository also has a cost. Developers can become over-whelmed with the number of reports submitted to the bug repository.

Each report is triaged to determine if it describes a valid problem and if so how the report should be categorized for handling through the development process. When developers are overwhelmed by reports, there are two effects. The first is that effort is redirected away from improving the product to managing the project. If a project gets thirty reports a day and it takes 5 min to triage a report then over two-person hours per day are spent triaging reports. If all of these reports lead to improvement in the code, this might be satisfactory cost to the project. However, for some projects, less than half of submitted reports lead to code improvements. For example, it was found that the Eclipse project had 5515

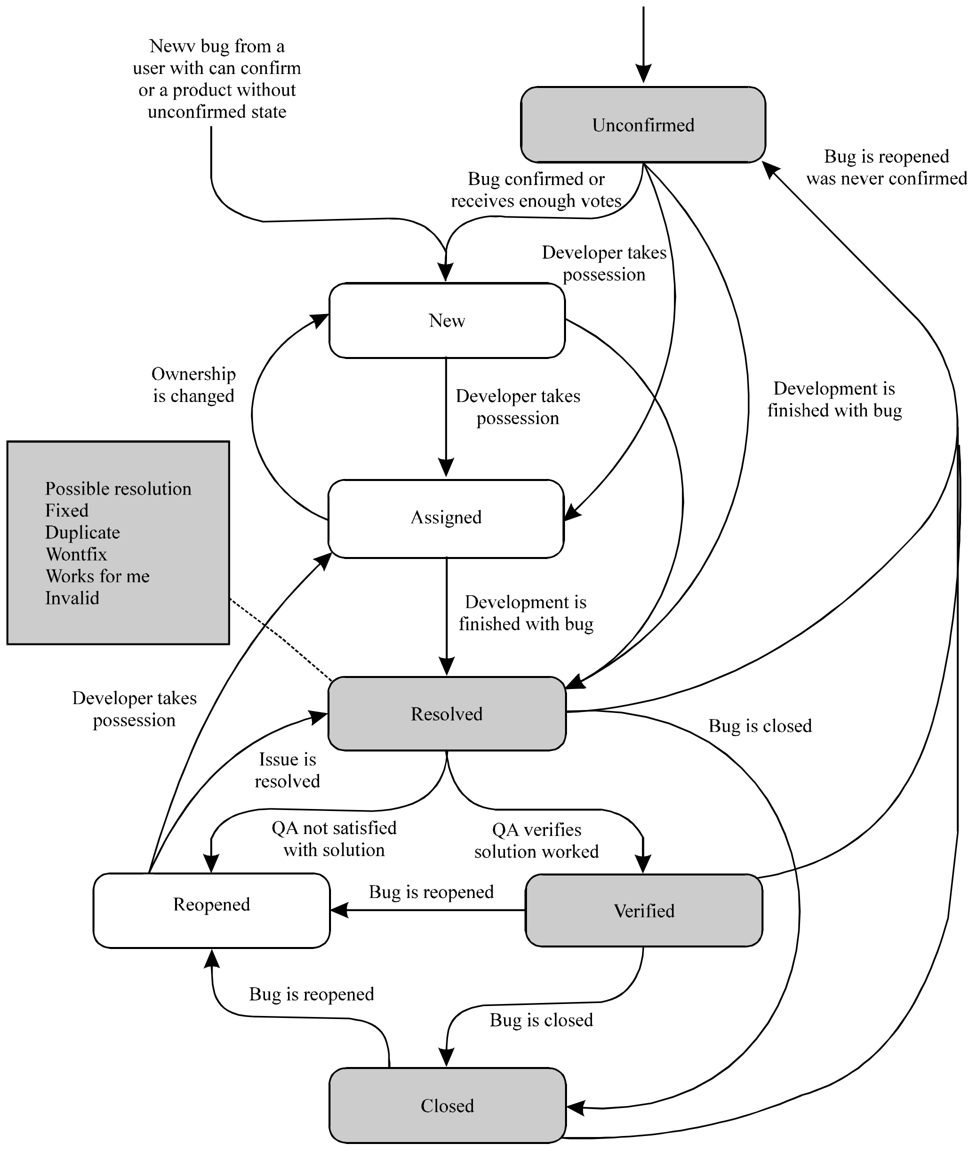
Newv bug from a user with can confirm or a product without unconfirmed state
Unconfirmed
Bug is reopened was never confirmed
Bug confirmed or receives enough votes
Developer takes possession
New
Ownership is changed
Developer takes possession
Development is finished with bug
Possible resolution
Fixed
Duplicate
Wontfix
Works for me
Invalid
Assigned
Development is finished with bug
Developer takes possession
Resolved
Bug is closed
Issue is resolved
QA not satisfied with solution
QA verifies solution worked
Reopened
Bug is reopened
Verified
Bug is reopened
Bug is closed
Closed

Fig. 1: Life cycle of a bug (Wang *et al*., 2008)

unproductive reports in 2004 (Alipour *et al*., 2013). The second effect is that reports may not be addressed in a timely fashion. If the number of reports that enter the repository is more than can be reasonably triaged within a suitable amount of time for the project then some reports may languish in the repository as other reports demanding more immediate attention take precedence. For an open-source project where the responsiveness of the development team to the community is often measured by how quickly reports are addressed and the number of outstanding reports, the rate at which reports are triaged can be an important factor in determining how well the project refinement is taking place. For example, Crowston *et al*., Berry and Castellanos, found that a measure of success for an open source project is the rate that users submitted bug reports and participated in project mailing lists. The person who triages the report, should have two goals. The first goal is to have the repository contain the smallest set of best reports for the project. The smallest set of best reports is desirable because reports typically enter the repository from various sources such as members of a technical support division, other developers and the user community (Fig. 1). Unfortunately with so many diverse sources of reports, some of the reports are not meaningful. For example, on a large project with many team members, several developers may submit a report describing the same bug. These duplicate reports need to be gathered together, so that, development effort is not wasted by having two developers solve the same problem. A triager also needs to filter reports that do not adequately enable a bug to be reproduced or that describe a problem whose cause is not the product but somewhat which is beyond the control of the developers such as the operating system. Sometimes, a triager also needs to filter the reports that are spam.

MATERIALS AND METHODS

Bug report duplication: Addressing these bugs frequently accounts for the majority of effort spent in the maintenance phase of a software project's life-cycle. This is why, researchers have been trying to enhance the bug-tracking systems to facilitate the bug-fixing process (Zimmermann *et al*., 2009; Sun *et al*., 2011). For several reasons, such as lack of motivation of users and defects in the search engine of the bug-tracking systems (Sun *et al*., 2011), the users of software systems may report some bugs that already exist in the bug-tracking system. These bug reports are called "Duplicates". The word duplicate may also represent the bug reports referring to different bugs in the system that are caused by the same software defect. Researchers have addressed several reasons for duplicate bug reports (Sun *et al*., 2011); Inexperienced users, poor search feature of bug-trackers and intentional/accidental re-submissions for reporting duplicate bugs, etc.

Hence, there is always need for inspection to detect whether a newly reported defect has been reported before. If the incoming report is a new bug then it should be assigned to the responsible developer and if the bug report is a duplicate, the report will be classified as a duplicate and attached to the original "Master" report. This process is referred to as triaging.

Identifying duplicate bug reports is of great importance, since, it can save time and effort of developers. Recently, many researchers like Bettenburg *et al*. (Sun *et al*., 2011) have focused on this problem. Here are some of the important motivations for detecting duplicate bug reports:

- Duplicate bug reports may be assigned to different developers by mistake which results
- In addition when a bug report gets fixed, addressing the duplicates as independent defects is a waste of time
- Finally, identifying duplicate bug reports can also be helpful in fixing the bugs, since, some of the bug reports may provide more useful descriptions than their duplicates (Sun *et al*., 2011)

When the number of daily reported bugs for a popular software is taken into consideration, manually triaging takes a significant amount of time and the results are unlikely to be complete. For instance in Eclipse, two person-hours are daily being spent on bug triaging (Zimmermann *et al*., 2009). Also, Mozilla reported in 2005 that :everyday", almost 300 bugs appear that need triaging" (Lazar *et al*., 2014).

A number of studies have attempted to address this issue by automating bug-report deduplication. To that end, various bug-report similarity measurements have been proposed, concentrating primarily on the textual features of the bug reports and utilizing Natural Language Processing (NLP) techniques to do textual comparison (Lazar *et al*., 2014). Some of these studies also exploit categorical features extracted from the basic properties of the bug reports (i.e., component, version, priority, etc.) (Sun *et al*., 2011). Some of these studies result in a method that automatically filters duplicate reports from reaching triagers. While, some other techniques provide a list of similar bug reports to each incoming report. Accordingly, rather than checking against the entire collection of bug reports the triager could first inspect the top-k most similar bug reports returned by this method (Sun *et al*., 2011).

RESULTS AND DISCUSSION

Dataset: As mentioned earlier, four large bug repositories are used in this study. These include: Android, Eclipse, Mozilla and OpenOffice bug repositories. Android is a Linux-based operating system with several sub-projects. The Android bug repository used in this study involves Android bug reports submitted from November 2007- September 2012. After filtering unusable bug reports (the bug reports without necessary feature values such as bug ID), the total number of bug reports is 37536 and 1361 of them are marked as duplicate. The Eclipse, Mozilla and OpenOffice bug repositories utilized in this study are adapted from Sun *et al*. (2011). Eclipse is a popular open source integrated development environment. It can be used to develop applications in Java and some other languages eclipse bug repository includes the bug reports reported in year 2008. After filtering unusable bug reports, the total number of reports is 43729 and 2834 of them are marked as duplicate. Open office is a well-known open source rich text editor. Open office contains several sub-projects including a word processor (writer), a spreadsheet (Calc), a presentation application (Impress), a drawing application (draw), a formula editor (Math) and a database management application. The open office bug repository includes 29455 bug reports in which there are 2779 bug

Table 1: Field of a hug report

Bug Id: ID of the bug report	(Issues) title: title of the bug report
Priority: The priority denotes how soon the bug should be fixed. This attribute typically varies between P1 to P5 where P1 denotes the highest priority	Resolution: This attribute indicates what happened to this bug. The value of this attribute varies between fixed, invalid, wontfix, duplicate, worksforme, incomplete
Type: The type of the report: defect, enhancement, task, feature	Company: The company bug occurred
Component: the relevant subsystem of the product for the reported bug	Severity: the impact of the bug on the software system. This attribute varies between trivial, minor, normal, major, critical and blocker
Product: the particular software application the bug is related to	Summary: A brief description of the problem
Open date: Date at which report is submitted	Description: A detailed declaration of the problem which may include reproduction steps and stack traces
Bug status: The attribute indicates the current state Of a bug. The value of this attribute varies between unconfirmed, new, assigned, reopened, ready, resolved, verified	Merge ID: If the report is a duplicate report, this field shows the ID of the report which the bug report is duplicating
cc: Users who are interested in the progress of this bug	Close date: Date at which report is closed
Operating system (op-sys): the operating system against which the bug is reported	Assigned to: the identifier of the developer who got assigned the bug
Short_desc: a one-line summary describing the bug	Version: the version of the product the bug was found in
Hardware: The hardware that the operating system is installed on it	Attachment: The attached file such as an bug image file or an bug report

bug reports marked as duplicate. Mozilla is a free software community best known for producing the Firefox web browser. In addition, Mozilla produces Thunder bird, Firefox Mobile and Bugzilla. The Mozilla bug repository exploited in this study contains 71292 bug reports (after filtering junk bug reports) in which 6049 of them are marked as duplicate (Anvik *et al*., 2006). Following are some facts on the volume of bug reports submitted to defect tracking systems: 13.016 bug reports were filed from Jun 4-5 between the release of eclipse platform version 3.0 and 3.1; Averaging 37 reports per day with a maximum of 220 reports in a single day. Number of bugs reported for Mozilla project = 51.154 during the period 2002-2006.

Fields of datasets: As indicated in Table 1, the bug reports exploited include the following features: description, summary, status, component, priority, type, version, product and etc, ..., According to need, these systems can even add custom information to the software system in which user data are received and stored. This information is used to detect duplicate bug reports. For example the status feature can have different values including “Duplicate” which means the bug report is recognized as a duplicate report by the triager.

Features extraction: Bug repositories are the systems which manages the bug reports sent by a broad community of users. Usually, the users have different knowledge, skill and vocabulary level to formulate report about a bug. Consequently, a bug tracking system is full of reports many of which are duplicates of each other and bug triaging is time consuming and error prone. Triagers can become overwhelmed by the number of reports added to the repository. Time spent in triaging also typically diverts valuable resources away from the improvement of the product to the managing of the development process.

According to the types of data in the bug reports is quite clear that these data are important to determine the similarity between bug reports. For this purpose, methods of data mining, text mining, information retrieval and natural language processing in the field to determine similarity measurements among the bug reports. Given that there are several ways to extract features and combines text has a number of features are extracted from the text reports.

Types of features: In terms of performance, features extracted from data set divided into four distinct categories: categorical features, text features, semantic features and structural features. This features following explanation will be given.

Categorical features: To compare the categorical features of a pair of bug reports. This measure the similarity between them based on their basic features (component, type, priority, product and version) indicated in Table 1, duplicate bug reports have similar categorical features. This motivates the use of categorical features in bug-deduplication.

In the following indicates the textual and categorical similarity measurement formulas applied in types of methods. These formulas are adapted from Sun *et al*. (2011) research’s. Here d, q are each about to send an bug report. In general it can be said that categorical features using attributes in the database table is extracted:

$$\text{Feature}_1(d,q)=\begin{cases}1, \text{ if } d.\text{prod}=q.\text{prod}\\0, \text{ otherwise}\end{cases}$$

$$\text{Feature}_2(d,q)=\begin{cases}1, \text{ if } d.\text{comp}=q.\text{comp}\\0, \text{ otherwise}\end{cases}$$

$$\text{Feature}_3(d, q) = \begin{cases} 1, \text{ if } d.\text{type} = q.\text{type} \\ 0, \text{ otherwise} \end{cases}$$

$$\text{Feature}_4(d, q) = \frac{1}{1+|d.\text{prio} - q.\text{prio}|}$$

$$\text{Feature}_5(d, q) = \frac{1}{1+|d.\text{vers} - q.\text{vers}|}$$

Textual features: The exact number of words and same terms calculate with using textual similarity metrics and graph clustering algorithms to determine the duplication. In order to accurately, most of these features are usually redundant and widely used conjunctions and words are deleted, so, as to prevent the creation of similar meaning between the reported bugs. In following are a few features extracted from the data bug are reported. BM25 is a measure of the similarity of text and d, q are each about to send a bug report (Sun *et al*., 2011):

$$\text{Feature}_1(d, q) = \text{BM25}_{ext}(d, q) // \text{of unigrams}$$

$$\text{Feature}_2(d, q) = \text{BM25}_{ext}(d, q) // \text{of biograms}$$

BM25F to compare long queries such as bug reports descriptions. This metric is based on TF-IDF weighting technique. BM25F is designed for short queries which usually have no duplicate words. For example, the queries in search engines usually include fewer than ten distinct words. However in the context of duplicate bug report retrieval, each query is a bug report. The query is structured such that it contains a short summary and a long description and it can sometimes be very long:

$$\text{BM25F}_{ext}(d, q) = \sum_{t \in d \cap q} \text{IDF}(t) \times \frac{\text{TF}_D(d, t)}{K_1 + \text{TF}_D(d, t)} \times W_Q$$

In equation above for each shared term t between a document d and a query q, the following components are calculated: One is the TFD (d, t) of a term t in a document d which is the aggregation of the importance of t in each textual field of d. Another one is WQ that involves weight from the query computed by TFQ (q, t). The last one is the IDF (t) which has an inverse relationship with the frequency of a term t across all the documents in the repository.

Feature engineering and extraction can be done to find the similarity (or dis-similarity) between two reports. Limited features make it hard to differentiate between two contrasting datasets; Pairs that are duplicates and pairs that are non-duplicates. Hence, a rich enough feature set is needed to make duplicate bug report retrieval more accurate. Textual similarity can be given in terms of features.

Table 2: Category in the software domain

Categories	Software domain
1	Efficiency
2	Functionality
3	Maintainability
4	Portability
5	Reliability
6	Usability

Semantic features: Feature extraction and similarity computation (semantic and lexical) between two bug reports is central to the problem of duplicate detection. Extracting discriminatory features and important indicators from bug description is key to the performance of any duplicate bug report detection system.

Structure features: A bug report is possible containing information may also call stack methods to accurately determine which part of the application bug occurred and this bug has occurred during what calling series. Structural features to get patterns are frequent in the call stack, so in terms of the similarities between the pattern of call stack to achieve the similarity of the two reports.

Contextual features: These features are usually trying to get the topic and concept of a report. For example are prepared a dictionary of word related to features of software quality such as: efficiency, functionality, maintainability, portability reliability and usability and review the use of these words in the context report and on what context it is determined that the bug report related software. Key words related to the software field has been shown in Table 2. Considering the dictionary is created, it can be achieved in a context similar reports.

Each contextual data-set adds some new contextual features to each bug report. The number of these contextual features is equal to the number of word lists included in the contextual data-set. These contextual word lists elaborate the raw data in the primitive bug reports before being used for the bug report deduplication process. The contextual word lists are showed below (Lazar *et al*., 2014):

- Architecture words
- Non-Functional Requirement (NFR) words
- LDA topic words
- Random English words
- General software engineering
- Android development

- Eclipse documentation
- Open Office documentation
- Mozilla documentation
- Labelled LDA

Related work: In this study, we review existing work on modeling bug data, bug triage and the quality of bug data with defect prediction.

In Ahmed Lamkanfi (Runeson *et al*., 2007) study's they propose the eclipse and mozilla defect tracking dataset, a representative database of bug data, filtered to contain only genuine defects (i.e., no feature requests) and designed to cover the whole bug-triage life cycle (i.e., store all intermediate actions). Our have used this dataset ourselves for predicting bug severity for studying bug fixing time and for identifying erroneously assigned components. Sharing these data with the rest of the community will allow for reproducibility, validation and comparison of the results obtained in bug-report analyses and experiments.

Jalbert and Weimer (2008) have introduced a classifier for incoming bug reports which combines the categorical features of the reports, textual similarity metrics and graph clustering algorithms to identify duplicates. In this method, bug reports are filtered based on an automatic approach. Their method is evaluated on a data-set of 29000 bugs from Mozilla Firefox. As a result, development cost was reduced by filtering out 8% of duplicate bug reports.

Wang *et al*. (2008) and Nguyen *et al*. (2012) used natural language information accompanied by execution information to detect duplicate bugs, evaluated on the Firefox and Eclipse bug repositories. Reports are divided into three groups: Run-time errors, feature requests and patch errors. They achieve better performance than relying solely on natural language information. This approach shows some promise behind using contextual information Lotufo *et al*. (Ma *et al*., 2007) studied how a triager reads and navigates through a bug and made a bug summarizer using this research. They successfully evaluated the quality of their summarizer on a wide survey of developers.

Ashish Sureka study's (Sun *et al*., 2011) presents an approach to compute text similarity between two bug reports to assist a Triager in the task of duplicate bug report detection. The central idea behind the proposed approach is the application of character n-grams as low-level features to represent the title and detailed description of a bug report. The advantages of the approach are language independence as it does not require language specific pre-processing and ability to capture sub-word features which is useful in situations requiring comparison of noisy text. The approach is evaluated on a bug database consisting of more than 200,000 bug reports from open source Eclipse project. The recall rate for the Top 50 results is 33.92% for 1100 randomly selected test cases and 61.94% for 2270 randomly selected test cases with a title to title similarity (between the master and the duplicate) of more than a pre-defined threshold of 50.

Research in duplicate report detection has primarily focused on word frequency based similarity measures paying little regard to the context or structure of the reporting language. Thus, in large repositories, reports describing different issues may be marked as duplicates due to the frequent use of common words. In Sean Banerjee (Zimmermann *et al*., 2009) study's, we present factor LCS, a methodology which utilizes common sequence matching for duplicate report detection. They demonstrate the approach by analyzing the complete Firefox bug repository up until March 2012 as well as a smaller subset of Eclipse dataset from January 1, 2008-December 31, 2008. They achieve a duplicate recall rate above 70% with Firefox which exceeds the results reported on smaller subsets of the same repository.

By Alipour *et al*. (2013) study's, they have exploited the domain knowledge and context of software development to find duplicate bug reports By improving bug deduplication performance companies can save money and effort spent on bug triage and duplicate bug finding. They use contextual word lists to address the ambiguity of synonymous software-related words within bug reports written by users who have different vocabularies. They replicated (Sun *et al*., 2011) method of textual and categorical comparison and extended it by adding their contextual similarity measurement approach. They have utilized the contexts of Android architecture, Non-Functional Requirements (NFRs) and the Android LDA-extracted topics (extracted by LDA and Labeled-LDA). By including the overlap of context as features they found that our contextual approach improves the accuracy of bug-report deduplication by 11.55% over Sun *et al*. (2011) method. This implies that by addressing the context of software engineering and relying on prior knowledge of software development they can boost bug de-duplication performance they conclude that to improve duplicate bug-report detection performance one should consider and not ignore, the domain and context of software engineering and software development.

Bug reports typically comprise a problem description in natural language text and often, structural elements such as patches, stack traces and source code. Research to date using of bug reports have treated all contents as natural language text but research can potentially benefit from treating such elements differently. In Bettenburg and etc study's developed a tool, infoZilla that extracts these elements from the reports with near perfect accuracy, as demonstrated by their evaluation of 800 ECLIPSE bug reports. Access to such piecewise elements from bug reports opens doors to several possibilities for research, for example, assignment of bug reports to developers and detection of duplicates and more.

In general, researches done in the field of detection duplicate bug reports can be categorized in four areas that in the following explained.

Information Retrieval (IR) techniques: Information retrieval is the activity of obtaining the needed information from a collection of information resources. IR techniques are applied on a broad spectrum of different scopes from image retrieval to web search. Here, we indicate some of the most frequently used IR techniques. Vector Space Model (VSM) is one of the tools exploited repeatedly in information retrieval. This model is commonly utilized for the purpose of comparing textual queries or documents. One of the outstanding methods of forming a weight-vector out of a text is the Term Frequency-Inverse Document Frequency (TF-IDF). TF-IDF is a weighting factor which denotes how important a word is to a document in a repository of documents. The basic formulas for the TF-IDF are as follows:

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

$$\text{idf}(t, d) = \log\frac{|D|}{|\{d;D;t \in d\}|}$$

$$\text{tf}-\text{if d}(t,d,D) = \text{tf}(t,f)*\text{idf}(t,d)$$

Where:

f (t, d)	= The frequency of the term t in the document d
idf (t, D)	= Shows if the term t is common across the documents
idf (t, D)	= Divides the total number of the documents by the number of documents containing term t

Information retrieval techniques are frequently applied to resolve the software engineering problems. These techniques pertain to the maintenance and evolution phases of the software life-cycle. These techniques are exploited for variant issues including feature/concept location, fault prediction, developer identification, comprehension, impact analysis, traceability links and refactoring.

Stack traces techniques: In this approach, for each incoming bug report, two different similarities are calculated between this report and all the existing ones. The first similarity metric is the Natural-Language-based Similarity (NL-S) in which the summary and description of the bug reports are converted to weight vectors using TF-IDF and compared with each other using cosine similarity metric. The second one is called Execution-information-based Similarities (E-S) in which a vector space model is used to calculate the similarity of the bug reports, based on the execution information. However in this similarity measurement, only the methods that are invoked during the run are studied without considering how many times each method has been invoked. Also, the canonical signature of each method is counted as one index term. Thus, the weight vectors for the execution information are created using TF-IDF and the similarities are measured by the cosine similarity metric. Finally, a combination of NLS and E-S contribute in ranking the most similar reports to a particular incoming bug report. The experimental result indicates that this approach is able to detect 67-93% of duplicate bug reports in the Firefox bug repository (Banerjee *et al*., 2012).

Textual and categorical similarity techniques: Jalbert and Weimer (2008) and Runeson *et al*. (2007) have proposed a technique that automatically classifies and filters arriving duplicate bug reports to save triager's time. Their classifier combines the surface features duplicate bug reports to save triager's time. Their classifier combines the surface features of the bug reports (non-textual features such as severity, operating system and number of associated patches), textual similarity measurements and graph clustering algorithms to identify duplicate bug reports. This classifier applies a linear regression over the features of the bug reports. Each document is represented by a vector in which each vector is weighted utilizing the following formula $W_i = 3+2 \log_2 (\text{freq})$ in which the W_i is the weight of word i in the document and freq is the count of word i in the document. The textual similarity between every two documents is calculated by the cosine similarity metric. The result of this similarity measurement is the basis for inducing a similarity graph. And a clustering algorithm is applied on the graph. Finally, the surface features are exploited to identify the duplicate reports. The experiments are performed on a subset of Mozilla bug reports. As the authors report, this approach can detect and filter 8% of duplicate reports automatically:

$$\text{Cosine}_{\text{sin}} = \frac{\sum_{i=1}^{n} C1_i \times C2_i}{\sqrt{\sum_{i=1}^{n}(C1_i)^2} \times \sqrt{\sum_{i=1}^{n}(C1_i)^2}}$$

In this formula, n is the number of word lists of the contextual data which is equal to the number of contextual features added to each bug report. $C1_i$ and $C2_i$ are the ith contextual features added to the first and second bug reports in the pair, respectively.

Topic model techniques: Nguyen *et al*. (2012) have proposed a novel technique called DBTM in which both IR-based techniques and topic extraction ones are applied to detect duplicate bug reports. To train the DBTM, the existing bug reports in the repository and their duplication information is utilized. For prediction, DBTM is applied to a new bug report and uses the train parameters to estimate the similarity between the bug report and existing reports in terms of textual features and topics. They have also proposed a novel LDA-based technique called T-Model to extract the topics from the bug reports. The T-Model is trained in train phase in a way that the words in bug reports and the duplication relation among them are used to estimate the topics, the topic properties and the local topic properties. In the prediction phase for any new bug report b_{new} the T-Model takes advantage of the trained parameters to find the groups of duplicates G that have the most similarity with b_{new} in terms of topics. This similarity is measured using the following formula: In which topicsim (b_{new}, b_i) is the topic proportion similarity between the bug reports b_{new} and b_i:

$$\text{Topicsim}(b_{new}.G) = \max_{b_i \in G}(\text{topicsim}(b_{new}, b_i))$$

In the study cited to measure the textual similarity between the bug reports, BM25F method (Sun *et al*., 2011) is exploited. To combine topic-based and textual metrics a machine learning technique called ensemble averaging is applied. Below, you can find the equation for calculating y which is the linear combination of the two above mentioned metrics:

$$y = \alpha \times y_1 + \alpha \times y_2$$

In the above function, y_1 and y_2 are textual and topic-based metrics. Also, $\propto_1$ and α_2 control the significance of these metrics in the duplicate bug report identification process. These factors satisfy $\propto_1 + \alpha_2 = 1$. This approach provides a list of top-K similar bug reports for every new report. The authors have performed their experiments on Open Office, Eclipse and Mozilla project bug repositories. And reported 20% improvement in the accuracy over the state-of-the-art.

The bug report deduplication approaches reviewed in this section could be divided into four groups. These groups are illustrated in Table 3.

Table 3: Related literature on detecting duplicate bug reports

Variables/Article title	Comparison technique	Retrieval technique	Evaluation metrics
Approaches applying IR techniques exclusively			
Detection of duplicate defect reports using natural language processing (Lazar *et al*., 2014)	Applying vector space model and cosine similarity metric.similarity metric considering the time frames	List of candidate duplicates	Recall rate
A discriminative model approach for accurate duplicate bug report retrieval (Nguyen *et al*., 2012)	Applying SVM to predict duplicates based on textual comparison metrics	List of candidate duplicates	Recall rate
Weight similarity measurement model based, object oriented approach for bug databases mining to detect similar and duplicate bugs [36](Ma *et al*., 2007)	Applying vector space model and cosine similarity metric to specify duplicates based on a specific threshold	Automatic filtering	Recall and precision
Detecting duplicate bug report using character n--gram-based features.(Sun *et al*., 2011)	Constructing the character ngrams of description and title of the reports and comparing them based on the number of shared character n-grams	List of candidate duplicates	Recall rate
Assisted detection of duplicate bug reports (Anvik *et al*., 2006)	Applying vector space model, cosine similarity metric and clustering to identify duplicates based on a specific threshold	List of candidate duplicates	Recall and precision
Stack traces based approaches			
An approach to detecting duplic ate bug reports using natural language and execution information (Banerjee *et al*., 2012)	Comparing bug reports textually using TF-IDF and cosine similarity metrics as well as execution information and combining these metrics	List of candidate duplicates	Recall rate
Textual and categorical similarity based approaches			
Automated duplicate detection for bug tracking systems (Runeson *et al*., 2007)	Applying vector space model, cosine similarity metric, using surface features and clustering the bug reports.	List of candidate duplicates	Recall rate and Area Under the ROC Curve (AUC)
Towards more accurate retrieval of duplicate bug reports (Sun *et al*. 2011; Anvik *et al*., 2005)	Applying a set of 7 comparisons including BM25F and categorical similarity metrics	List of candidate duplicates	Recall rate and Mean Reciprocal Rank (MRR)
Topic model based approaches			
Duplicate bug report detection with a combination of information retrieval and topic modeling (Jalbert and Weimer, 2008)	Applying BM25F and LDA based topics extraction similarity metric and combining the metrics using ensembled averaging	List of candidate duplicates	Recall rate

CONCLUSION

Finally, a triager may indicate that the problem will not be fixed or that the feature will not be added to the product. Reports meeting any of these criterion must be identified, so that, development effort can focus on the reports that lead to product improvements. For example, nearly a third of the reports submitted to the Firefox project created between May 2003 and August 2005 were marked as duplicates (Banerjee *et al*., 2012). One can call triage decisions that result in a report being designated as not meaningful as a repository-oriented decisions. Generally, automatic detection of duplicate defect reports and linking similar defect reports is a technically challenging problem due to the following reasons:

- Bug reports are expressed in natural language text. Natural language is vast and ambiguous
- The quantity of problem reports in large and complex software setting is huge
- Presence of poorly expressed bug reports (missing information, noisy text) poses additional technical challenges

REFERENCES

Alipour, A., A. Hindle and E. Stroulia, 2013. A contextual approach towards more accurate duplicate bug report detection. Proceedings of the IEEE 2013 10th International Working Conference on Mining Software Repositories (MSR), May 18-19, 2013, IEEE, San Francisco, California, USA., ISBN:978-1-4673-2936-1, pp: 183-192.

Anvik, J., L. Hiew and G.C. Murphy, 2005. Coping with an open bug repository. Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange, October 16-17, 2005, ACM, New York, USA., ISBN:1-59593-342-5, pp: 35-39.

Anvik, J., L. Hiew and G.C. Murphy, 2006. Who should fix this bug?. Proceedings of the 28th International Conference on Software Engineering, May 20-28, 2006, ACM, New York, USA., ISBN:1-59593-375-1, pp: 361-370.

Banerjee, S., B. Cukic and D. Adjeroh, 2012. Automated duplicate bug report classification using subsequence matching. Proceedings of the 2012 IEEE 14thInternational Symposium on High-Assurance Systems Engineering, October 24-27, 2012, IEEE, Omaha, Nebraska, ISBN:978-1-4673-4742-6, pp: 74-81.

Jalbert, N. and W. Weimer, 2008. Automated duplicate detection for bug tracking systems. Proceedings of the IEEE International Conference on Dependable Systems and Networks with FTCS and DCC, June 24-27, 2008, IEEE, Anchorage, Alaska, ISBN: 978-1-4244-2397-2, pp: 52-61.

Lazar, A., S. Ritchey and B. Sharif, 2014. Improving the accuracy of duplicate bug report detection using textual similarity measures. Proceedings of the 11th International Working Conference on Mining Software Repositories, May 31-June 1, 2014, ACM, New York, USA., ISBN: 978-1-4503-2863-0, pp: 308-311.

Ma, Y., S. Lao, E. Takikawa and M. Kawade, 2007. Discriminant analysis in correlation similarity measure space. Proceedings of the 24thInternational Conference on Machine Learning, June 20-24, 2007, ACM, New York, USA., ISBN:978-1-59593-793-3, pp: 577-584.

Nguyen, A.T., T.T. Nguyen, T.N. Nguyen, D. Lo and C. Sun, 2012. Duplicate bug report detection with a combination of information retrieval and topic modeling. Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, September 3-7, 2012, ACM, New York, USA., ISBN: 978-1-4503-1204-2, pp: 70-79.

Runeson, P., M. Alexanderson and O. Nyholm, 2007. Detection of duplicate defect reports using natural language processing. Proceedings of the 29th International Conference on Software Engineering, May 20-26, 2007, Minneapolis, MN., pp: 499 -510.

Sun, C., D. Lo, S.C. Khoo and J. Jiang, 2011. Towards more accurate retrieval of duplicate bug reports. Proceedings of the 2011 26thIEEE/ACM International Conference on Automated Software Engineering, November 6-10, 2011, IEEE Computer Society, Washington, DC., USA., ISBN:978-1-4577-1638-6, pp: 253-262.

Wang, X., L. Zhang, T. Xie, J. Anvik and J. Sun, 2008. An approach to detecting duplicate bug reports using natural language and execution information. Proceedings of the 30thInternational Conference on Software Engineering, May 10-18, 2008, ACM, New York, USA., ISBN:978-1-60558-079-1, pp: 461-470.

Zimmermann, T., R. Premraj, J. Sillito and S. Breu, 2009. Improving bug tracking systems. Proceedings of the 31st International Conference on Software Engineering-Companion ICSE-Companion, May 16-24, 2009, IEEE, Vancouver, Canada, ISBN:978-1-4244-3495-4, pp: 247-250.