A Bug Report Analysis and Search Tool and Its Validation

Yguaratã Cerqueira Cavalcanti[1,3], Eduardo Santana de Almeida[2,3], Silvio Romero de Lemos Meira[1,3]

[1]Center for Informatics – Federal University of Pernambuco (UFPE)
[2]Computer Science Department – Federal University of Bahia (UFBA)
[3]RiSE – Reuse in Software Engineering (RiSE)
{ycc,srlm}@cin.ufpe.br, esa@dcc.ufba.br

Abstract. *According to recent work, duplicate bug report entries in bug trackers impact negatively on software maintenance and evolution productivity due to, among other factors, the increased time spent on report analysis. Such type of duplication is characterized by the submission of two or more bug reports that describe the same software issue. In this sense, this dissertation investigates and characterizes the problem of bug report duplication and proposes a solution to it.*

1. Introduction

Software maintenance and evolution are characterised by their high cost and slow speed of implementation. However, they are inevitable activities – almost all software that is useful and successful stimulates user-generated requests for change and improvements [7]. Sommerville [29] is even more emphatic and says that software changes is a fact of life for large software systems. In addition, a set of studies [12, 13, 17, 23] has stated along the years that software maintenance and evolution is the most expensive phase of software development, taking up to 90% of the total costs.

All of these characteristics from software maintenance leaded the academia and industry to investigate constantly new solutions to reduce costs in such phase. In this context, Software Configuration Management (SCM) is a set of activities and standards for managing and evolving software, defining how to record and process the proposed system changes, how to relate these to system components, among other procedures. For all these tasks, it has proposed different tools, such as version control systems and bug trackers [29]. However, some issues may arise due to these tools usage. In this work, the focus are the issues from bug trackers, as it will be discussed along this paper.

Aiming to improve change management processes of software projects development, some organizations have used specific systems (generally called *bug-trackers*) to manage, store and handle change requests (also known as *bug reports*). A bug report is defined as a software artifact that describes some defect, enhancement, change request, or an issue in general, that is submitted to a bug tracker; generally, bug report submitters are developers, users, or testers. Such systems are useful because changes to be made in a software can be quickly identified and submitted to the appropriate people [3].

Moreover, the use of bug trackers helps to monitor software evolution, because bug reports are recorded in a database as well as people involved in a particular bug report are recorded. Thus, changes and their respective responsible can be easily found. Organizations also use such systems to guide the software development, thus any task

to be undertaken in the software development process must be registered and monitored through a bug-tracker. Moreover, the historical data of these systems can be used as history and documentation for the software. Examples of such systems are Bugzilla, Mantis and Trac.

Each bug report is stored with a variety of fields of free text and custom fields defined according to the necessity of each project. In Trac, for example, it is defined fields for summary and detailed description of a bug report. In the same bug report it can also be recorded information about software version, dependencies with other bug reports (duplicate bug reports, for example), the person who will be assigned to the bug report, among other information. Moreover, during the life cycle of a bug report, comments can be inserted to help solving it.

Nevertheless, some challenges have emerged through the use of bug trackers, among them, we can cite: dynamic assignment of bug reports [4], change impact analysis and effort estimation [30], quality of bug report descriptions [20], software evolution and traceability [28], and duplicate bug reports detection [16]. These issues are further discussed later.

The focus of this work is trying to avoid duplicate bug reports submission; the duplicate problem is characterized by the submission of two or more bug reports that describe the same issue. In this context, this work investigates the problem of bug report duplication emerged by bug trackers, characterizing it empirically to understand its causes and consequences to the software development projects, and provides a tool for search and analysis of bug reports to reduce the effort spent on such tasks.

2. Mining Bug Report Repositories: A Brief Survey

There is a variety of work related to mining bug report repositories. However, the work found on the literature are relatively new, dating back from 2003. In general, these types of repositories have been mined for different purposes, such as: *bug reports similarity* (also referenced as duplicate detection), *dynamic assignment of bug reports*, *software evolution and traceability*, *change impact analysis and effort estimation*, and *quality of bug report descriptions*.

There are also studies that combine mining bug report repositories and mining other types of repositories, such as source code repositories. For more information about other work not described here, there is a taxonomy proposed by Kagdi *et al.* [19]. In addition, for more detailed characteristics about the work presented here, the complete text of this dissertation must be consulted.

Bug Reports Similarity (duplicate detection). Duplicate bug reports detection consists on searching for past bug reports to find similar bug reports that describe the same issue as the one being reported, in order to avoid duplicate submission. Podgurski *et al.* [25] was the first to investigate bug reports similarity. The bug reports explored in their work were software failures automatically submitted when the software did not work properly. Thus, Podgurski *et al.* proposed an automated support for classifying these reports in order to prioritize and diagnostic their causes.

The work performed by Hiew [16] is closer to ours than the first one described. It investigated the duplication problem caused by natural language bug reports submis-

sion. Hiew proposed to group similar bug reports into *centroids*, thus it would be possible to compare incoming bug reports to the *centroid* with high similarity through the Term Frequency-Inverse Document Frequency (TF-IDF) [5] technique. In this same direction, Runeson *et al.* [26] addressed the problem of detecting duplicated bug reports using Natural Language Processing (NLP) techniques.

In Wang *et al.* [31], it was proposed an approach to mitigate bug reports duplication problem using NLP and execution information. The execution information is concerned to data about the software execution when the error occurred, such as method calls or variables state. This type of data was combined with natural language data to improve the recall.

Bug Report Dynamic Assignment. The bug report dynamic assignment consists of identifying which is the best developer to solve a new bug report. Several work have used machine learn techniques combined with versioning system data and/or bug report repositories. Anvik *et al.* [4] presented an approach for semi-automated bug report assignment. The approach used a machine learning algorithm to a bug report repository to learn the types of bug reports that each developer resolves. The work of Canfora and Cerulo [10] also proposed a method to bug report assignment, however in such work it was used information from versioning systems combined with bug reports information.

Anvik and Murphy [2] compared whether versioning systems or bug report repositories is better to assign the best developer to a bug report. The work concluded that using bug report repositories is better if the objective is to determine the expertise group with less false positives (developers that are not expert in the given subject), while versioning systems are better for retrieving all experts for a given problem (in this case false positives can occur).

Software Evolution and Traceability. Mining bug repositories for software evolution and traceability is concerned with understanding what drives the changes performed in the software along the time. Sandusky *et al.* [28] conducted an empirical research about Bug Report Networks (BRNs) in open source projects. According to them, a BRN is created when members of a software development project assert duplication, dependency, or reference relationships among bug reports. They pointed that BRNs understanding can be useful for decreasing cognitive and organizational effort, refined representations of software and work-organization issues, and rearrange the relationships among project members.

Antoniol *et al.* [1] proposed a framework to merge information from bug report repositories, source code, and versioning systems. Such framework aids the developer to browse and navigate through the information provided by such artifacts in an interconnected way. For example, some developer could use the framework to visualize what bug reports were fixed in a given software version. Furthermore, he/she could visualize what files of source code were modified.

Koponen and Lintula [21] proposed an approach to integrate versioning systems and bug report repositories. Koponen and Lintula investigated if the changes in such projects were driven by bug reports. It was observed that in some projects only a small percentage of the changes is guided by bug reports, while in other the opposite is true, thus not having a confidence pattern.

There are also other work in the same direction of Koponen and Lintula [21], such as Kagdi *et al.* [18] and Fischer *et al.* [14, 15]. In the first, commits of versioning system were analyzed to verify frequent co-changes sets of artifacts (e.g. source code and documentation). The second one aimed to understand the software evolution by integrating versioning systems and bug report repositories, thus looking at the bug reports it is possible to determine which commits were performed to solve a specific issue.

Impact Analysis and Effort Estimation. The impact analysis and effort estimation purposes are related to determine the amount of time, costs and complexity that a bug report needs to be resolved. For that purpose of effort estimation, three work were found in the literature: Song *et al.* [30], Panjer [24], and Weiss *et al.* [32]. All of them used data from bug report repositories as input for their approach. For impact analysis there is the work of Canfora and Cerulo [9], where it was explored bug report repositories and versioning systems using information retrieval techniques to predict the impacted source files for a new bug report submission.

Bug reports Quality. The quality analysis of bug reports is concerned with how submitters are describing software issues on bug reports free-text fields. In Ko *et al.* [20], it was performed a study to understand how submitters describe software problems. They discovered that bug reports summary generally describe software entity or behavior and its execution context. Additionally, Bettenburg *et al.* [8] performed a survey with Eclipse developers to understand what type of information they use in bug reports and problems found.

3. The Bug Report Duplication Problem: A Characterization Study

This section presents a characterization study about bug repositories and search and analysis of bug reports. The Goal Question Metric (GQM) method [6] was used to define this characterization study. Briefly, the GQM consists of the definition of the study's goal, the questions to be answered, and the related metrics. Due to space constraints, we omitted some metrics explanation.

The goal of this study was to analyze *bug repositories* and the *activities for searching and analyzing bug reports* with the purpose of *understanding them* with respect to the *possible factors that could impact on the duplication problem and their consequences on software development*, from the point of view of the *researcher*, in the context of *software development projects*. Eight (8) open source projects[1] and one private project were selected. For the private project, we chose bug reports from a project being developed at Recife Center for Advanced Studies and Systems (C.E.S.A.R.)

Do the analyzed projects have a considerable amount of duplicate bug reports? This is the starting point of this study. Thus, we investigated the projects to find out if they have duplicate bug reports in their repositories, and if the amount of duplicates is large enough to cause problems. We answered it by computing the percentage of duplicate bug reports.

Is the submitters productivity being affected by the bug report duplication problem? The productivity here is measured in terms of time that is needed to perform bug tracking activities, such as search and analysis. To answer this question, we considered

[1] Bugzilla, Firefox, Eclipse, GCC, Thunderbird, Evolution, Tomcat, Epiphany

the amount of time spent to search and analyze bug reports before opening a new bug report, the ratio between the average time to resolve duplicate bug reports and average time to resolve valid bug reports, and the average frequency of bug reports per day.

Is there a common vocabulary for bug report descriptions? It is believed that a controlled vocabulary could help avoiding duplicate bug reports [22]. For example, with a well defined vocabulary, the submitters could perform better searches using keywords closer from those present in the new bug report. To answer it, we computed the percentage of common words shared in a bug report group to describe the same problem.

How are the relationships between master bug reports and duplicate bug reports characterized? In this work, we consider three types of bug reports: *unique*; *master*; and *duplicate*. We computed the distribution of each type. The following distributions were analyzed: master bug reports that have only one duplicate bug report, also known as *one-to-one* relationships; and master bug reports that have more than one duplicate bug report, also known as *one-to-many* relationships.

Does the type of bug report influence the amount of duplicates? There are mainly two types of bug reports: *enhancements* and *defects*. Intuitively, it may be argued that defects have more duplicates, because it is more likely that two users will notice the same defect than the same enhancement. For each bug report type we calculated the ratio between the duplicate and total bug reports.

What are the possible factors that could impact on the bug report duplication problem? We chose six variables that we believe could be the factors for duplication or, at least, have some indirect relation to it: `staff size` – the number of people involved in the project development; `number of submitters` – the amount of submitters in the period that we collected the bug reports; `software size` – the number of lines of code (LOC); `software life-time` – the time that a software project has been in development; `bug Repository size` – the amount of bug reports; `submitters' profile` – the type of submitter profile that is more susceptible to submitting duplicate bug reports (*sporadic*, *average*, and *frequent*).

3.1. Main Findings on The Bug Report Duplication Problem

This study was conducted with the goal of understanding which characteristics of the projects can be factors for bug report duplication. We would like to understand also if the problem actually exists in the examined projects, and where it impacts on development. After all the analysis performed with the outcome data from this study, we can point out the main findings as following:

a) All the analyzed projects have duplicate bug reports in their repositories, which we consider that they are being affected by the duplication problem. Some projects are less affected than others, but in general, all of them are affected;

b) The submitters productivity in the analyzed projects is being affected by the bug reports duplication problem. According to the estimative of this study (taking into account all projects), almost 48 man-hours are necessary each day to do search and analysis of bug reports due to duplication in bug repositories;

c) Generally, the submitters do not use a common vocabulary to describe the content of bug reports. This situation makes it more difficult to identify duplicates. Moreover,

none of the projects have explicitly defined a vocabulary to control the descriptions;

d) We observed that submitters from private project are more likely to submit reports using a common vocabulary than submitters from open source projects;

e) For the bug reports grouping analysis, we observed that more than 80% of the groups are composed by one-to-one grouping type. Thus, clustering and IA techniques may not be applicable to identify duplicates;

f) The bug report duplication problem can occur independently of the type of bug reports that are being submitted (for example, defect and enhancement bug reports);

g) The number of LOC does not seem to be a factor to duplication problem;

h) The size of the repository does not seem to be a factor for duplication;

i) Projects' life-time does not seem to be a factor for duplication;

j) The staff size does not seem to be a factor for the duplication problem;

k) The profile of the submitter is a determining factor for the submission of duplicates, as well as the amount of submitters; and

l) According to our study, sporadic submitters seem to be more likely to submit duplicate bug reports as well as a larger quantity of submitters can also increase the amount of duplicates.

4. BAST: Bug Report Analysis and Search Tool

In this section, we present a tool, its architecture and features, called Bug Report Analysis and Search Tool (BAST), built in order to facilitate the activities of search and analysis of bug reports, focused on the detection of duplicates bug reports submission. The BAST goal is to provide a Web-based application for search and analysis of bug reports, thus, the time spent in these activities can be reduced and more duplicates can be avoided.

4.1. BAST Search Features

The search for bug reports is one of the main and most important services that BAST provides for the submitters. It is through these searches that the submitters find similar bug reports in order to not submit duplicates. The following subsections describe the mechanisms for indexing and ranking, query, and visualization of searches.

Ranking and Indexing – Vector Space Model. To sort the results of a search, it was used the Vector Space Model (VSM) [5]. With VSM, the documents are represented in a space of vectors of terms. Each dimension (vector) of this space is represented by a term, and this term is associated with a weight known as *Term Frequency-Inverse Document Frequency* (TF-IDF) [27]. The symbol TF-IDF is an abbreviation for *Term Frequency-Inverse Document Frequency*, and means that the value of the weight of a term is calculated taking into account the frequency in which this term appears in the collection of documents, and the frequency in which it appears in each document.

Queries. In BAST, it is provided *natural language queries* with a combination of a variation of *boolean queries* Baeza-Yates and Ribeiro-Neto [5]. Such variation on boolean queries allows submitters to search for complete phrases. For example, if a submitter put some keywords enclosed in quotes (i.e. "*radio fm error*"), the system will

return only the bug reports that match the entire query "*radio fm error*", also respecting the order of the keywords. BAST also implements some search filters that can be inserted directly in queries, improving the use of keywords.

4.2. Architecture Components

The architecture of the tool is composed of a database module, a core module, and a Web interface module. Figure 1 illustrates a simplified organization of the architecture of the application: the database stores in an organised way the bug reports from the bug repositories to facilitate further searches; the main module has sub-modules for text processing, indexing of content, parsers, search, and information extractors; and the Web module implements the user interaction features that will be exposed to submitters through a Web browser. Next, it will be provided more details for each component.

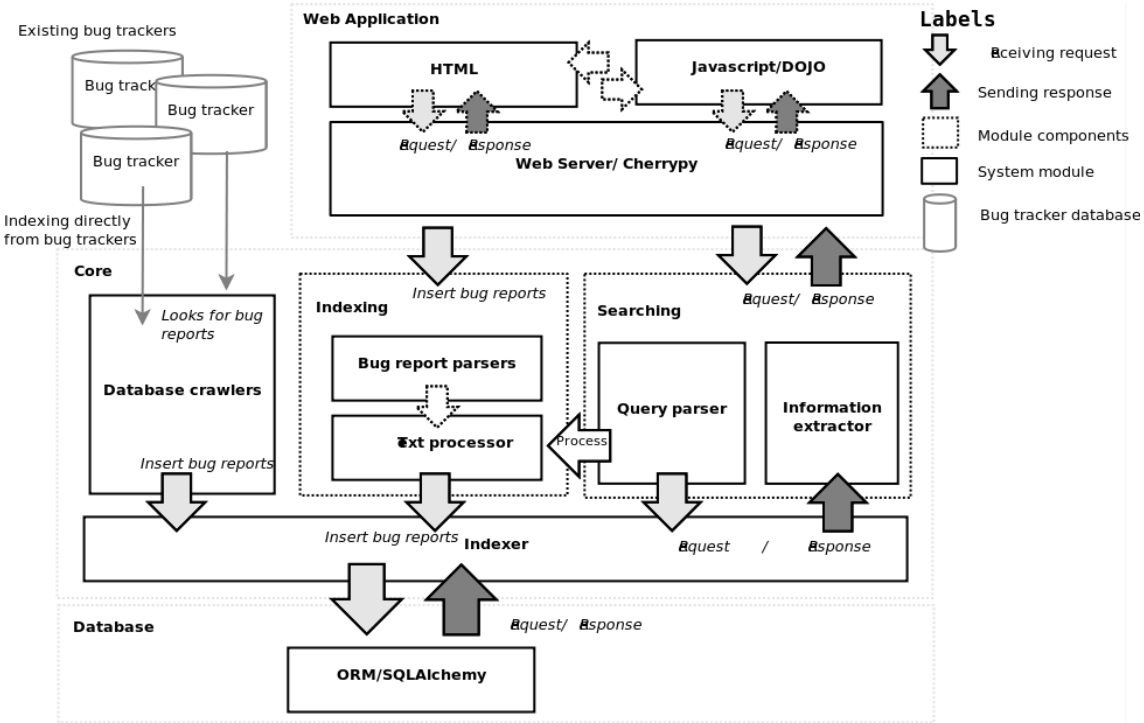

Figure 1. BAST Architecture - Web interface, core module and database

BAST Database. It uses the database to store all data from bug reports inserted in the application. We did not use only a simple structure for indexing and retrieval, as provided by Lucene[2], because BAST needs to make complex SQL queries, such as one to draw relationships among bug reports. The technologies used to implement the database were: *MySQL* to implement the database; and *SQLAlchemy* – which is an Object-Relational Mapper (ORM) for Python language.

BAST Web Application. Running the tool on the Web was one of the main requirements defined for the construction of the tool. To achieve such purpose, we chose the following tools: *HTML*, *DOJO JavaScript Toolkit* and *Cheetah* for the interface; and *Cherrypy* for the Web application.

[2]http://lucene.apache.org

BAST Core Components. BAST Core module contains the main components of the tool, which are responsible for running the search engine, indexing, information extraction and so on. These components are *database Crawlers*, *bug report Parsers*, *text processors*, *indexers*, *query parsers*, and *information extractors*.

5. Case Study at C.E.S.A.R.

In this section, it is described a case study conducted inside a private test center partner from C.E.S.A.R., where BAST was tested during 30 days by one tester during a real cycle for software testing. In this context, BAST was compared with the baseline tool from such organization. Although it was an initial evaluation for BAST, the results were very significant. Furthermore, this case study served as a pilot project for the experiment that will be described next.

The null and alternative hypothesis are as follows, respectivelly: H_0: $\mu_{time\ with\ BAST} >= \mu_{time\ with\ baseline}$ and $\mu_{duplicates\ avoided\ with\ BAST} <= \mu_{duplicates\ avoided\ with\ baseline}$; H_1: $\mu_{time\ with\ BAST} < \mu_{time\ with\ baseline}$ and $\mu_{duplicates\ avoided\ with\ BAST} > \mu_{duplicates\ avoided\ with\ baseline}$

Case study design. The submitter was instructed on how he should use both tools to search and analyze the bug reports. Thus, we divided the assessment period in two treatments: the first stage (from July 17 to August 07), the submitter should carry out the analysis first in the private organization internal tool, and if he did not find a similar bug report, he should use our tool to perform a new analysis; in the second stage (from August 08 to August 29), this sequence was reversed.

Quantitative analysis mechanisms and data gathering. Descriptive statistics, such as percentages, mean, standard deviation and pie and bar charts were used to analyze the results. For data gathering we used spread-sheets where the submitter was responsible for recording the types of bug reports that were analyzed, the time spent on each analysis and, when a duplicate is found, to specify in which tool it was found.

5.1. Result Analysis

During the case study, it were examined 144 bug reports by the *bug report master*. A description of the types of bug reports analyzed in each treatmen is described next: *Unreproducible* are errors that testers and developers are unable to reproduce; *Feature Not Yet Available* are bug reports about the addition of new features; *WAD* is an acronym for *Work as Design*; *H/W Issue* means that the error found is caused by a defect in the hardware; *Workaround* means that the errors found can be avoided if the tests run in another way.

Analysis of the First Treatment. During the first treatment, it was analyzed 42 bug reports. According to the analysis, 72% of bug reports that were analyzed would be duplicate if they were not avoided. Meanwhile, only 14% were valid bug reports, 7% were *Unreproducible*, 2% *WAD*, 2% *H/W Issue*, and 2% *Feature Not Yet Available*. These data show that the majority of time spent with analysis and search is due to duplicates.

Furthermore, performing the analysis activity with the baseline tool prevented the submission of 58% of duplicates, while BAST prevented 35%. Moreover, 7% were avoided due to the information provided by developers, through email or other type of communication. Therefore, BAST had lower performance than the baseline tool in order to prevent duplicates.

Another analysis was the average time spent on search and analysis using the baseline tool and BAST. Although BAST has avoided less duplicates than the baseline tool, as mentioned before, the time to do search and analysis with BAST was less than with the baseline tool.

Analysis of the Second Treatment. During the period of the second treatment, it was analyzed 99 bug reports. According to the analysis, 44% of bug reports were duplicates that had been avoided, while 34% were valid bug reports, 9% were *unreproducible*, 7% *WAD*, 2% *H/W Issue*, 2% *workaround* and 1% *Feature Not Yet Available*.

Although the amount of bug report submitted increased, there was a reduction on the submission of duplicate bug reports and a growth of valid bug reports submission. We believe it is a consequence of the project maturation; testers and developers are more engaged and there is better knowledge of the bug reports currently being handled.

In this second treatment, the bug reports were analyzed first using BAST, and if it was not found similar bug reports, a new analysis should be performed with the baseline tool. At the end of this treatment, the analysis made with BAST avoided the submission of 89% of duplicates, while the baseline tool prevented just 7%. In addition, 7% were avoided due to the fact that the submitter had prior knowledge of similar bug reports.

The average time spent on search and analysis of bug reports on both tools did not have big difference. Although BAST had a little better performance in the second treatment (it spend 1 minute less than baseline tool in average), in regard to the time of search and analysis, we believe that further analysis (i.e. to analyze the complexity of bug reports) should be done to decide whether such difference was caused by the use of BAST or if it was influenced by other factors.

Analysis Conclusion. Although the individual analysis of the treatments showed few difference among the tools in some aspects, such as the time spent on analysis, according to the descriptive statistics analysis we can confirm the alternative hypothesis μ_1 and refute the null hypothesis μ_0. In other words, BAST can save more time than the baseline tool during the analysis of bug reports, and it can also avoid more duplicates.

6. BAST Empirical Evaluation Experiment

A controlled experiment was performed with 18 subjects in order to evaluate the tool against a baseline tool, so that more concrete conclusion can be drawn. This section describes such experiment, discussing its definition, planning, operation, analysis and interpretation.

Definition. This experiment was also performed using the GQM method [6]. The *goal* of this experiment is to analyze *a tool to improve search and analysis of bug reports* for *evaluating it* with respect to its *effectiveness* and *efficiency* on detection of duplicate bug reports and time saving, in the *view point of* researchers, in the *context of* software development projects. The following *quantitative* and *qualitative* questions were defined:

Is there a reduction on the number of duplicated bug reports with the new tool?

Is there a reduction on the time that submitters spend to perform the search and analysis of bug reports with the tool adoption?

Did the submitters have difficulties to use the tool?

The experiment was performed as an *off-line experiment*. The subjects will be composed by M.Sc. *students* from the Computer Science department at Federal University of Pernambuco/Brazil. In addition, the experiment will be performed distributed, which means that the subjects are free to choose their work environment, such as their home or university laboratories.

6.1. Planning

The subjects of this experiment were selected by *convenience sampling* [33]. All subjects received descriptions of defects from some open source software project. However, it is important to highlight that such descriptions are not the bug reports themselves, but only few words describing software errors.

Thus, it was created two lists of such objects for the experiment, where each list contained 32 descriptions, being 50% with defects that already have bug reports describing them in the repository, and 50% with unique/not-reported defects. It is crucial that such list holds some descriptions about already submitted bug reports, thus we can determine which duplicates were correctly avoided.

Descriptive statistics was used to analyze the data from the experiment. In order to test the hypotheses defined for the experiment, it will be used a paired *test-t* [33]. The qualitative analysis was conducted to understand the subjective aspects of the tool, such as the difficulties faced by the subjects during the use of the tool. The subject was asked about the applicability and usability of the tool.

For the experiment design it was used the *one factor with two treatments* design [33]. In such case, the factor is BAST. Thus, there is a treatment with such tool and another with the baseline tool. The two treatments are described as follow: *Treatment 1* – in the first treatment, the group of subjects received the list of error descriptions. Then, before submitting such errors as new bug reports, they used BAST to perform searches and analysis of existing bug reports in order to avoid duplicates submission; in the *Treatment 2* the group of subjects used the baseline tool to perform the searches and analysis.

The null and alternative hypothesis were defined as fallows, respectively: H_0: $\mu_{time\ with\ BAST} >= \mu_{time\ with\ baseline}$ and $\mu_{duplicates\ avoided\ with\ BAST} <= \mu_{duplicates\ avoided\ with\ baseline}$; H_1: $\mu_{time\ with\ BAST} < \mu_{time\ with\ baseline}$ and $\mu_{duplicates\ avoided\ with\ BAST} > \mu_{duplicates\ avoided\ with\ baseline}$

6.2. Operation

One open source project was selected for the execution of this experiment, Firefox Internet browser. We chose this project because it is a tool well known both by end users and Computer Science students.

Baseline tool. The Bugzilla bug tracker with all bug reports from Firefox was chose to compose the baseline. Since Firefox uses Bugzilla to handle its bug reports, it was convenient to choose this bug tracker. Furthermore, Bugzilla is one of the most known bug trackers, being used by several open source and private projects.

Bug reports lists. It was delivered to the participants of the experiment two lists, each one with 32 descriptions of error of Firefox. To compose this amount, for each list it was randomly chose 16 (50% of the list) bug reports that were submitted to the Firefox project during the year 2008, and the other 50% were composed of unique bug reports.

6.3. Analysis and Interpretation

Table 1 shows the data obtained during the experiment. The first column shows the ID of the participant, second and third columns show the time spent in analysis of bug reports, and the remaining columns show the amount of duplicate bug reports avoided. We firstly used descriptive statistics to visualize the data collected.

	Time spent on analysis		Duplicates avoided	
#	BAST	Bugzilla	BAST	Bugzilla
1	3.09	2.56	5	9
2	3.03	2.47	8	10
3	3.31	3.09	8	12
4	6.78	6.84	13	10
5	5.1	4.82	4	2
6	3.06	2.75	11	11
7	4.97	3.91	12	9
8	5.04	9.56	2	8
9	5	2.97	8	8

	Time spent on analysis		Duplicates avoided	
#	BAST	Bugzilla	BAST	Bugzilla
10	3.63	3	7	10
11	6.84	6.88	7	8
12	1.78	2.66	6	4
13	6.66	5.41	9	10
14	3.69	4.19	13	10
15	6.47	4.31	9	11
16	3.75	2.72	8	10
17	4.47	4.91	6	8
18	4.97	4.78	0	0

Table 1. Collected data during the experiment.

Descriptive statistics Table 2 shows some statistics about the experiment results. For time spent on analysis, BAST had mean value of 4.54 minutes and standard deviation (SD) of 1.49, while Bugzilla had mean value of 4.32 and SD of 1.91. The differences among these values are few, thus we plot boxes to better understand them. From Figure 2, we can conclude that people using Bugzilla to analyze bug report spent a little less time than using BAST. Furthermore, it is important to note that most of subjects from BAST keep their time spent on analysis below the median value.

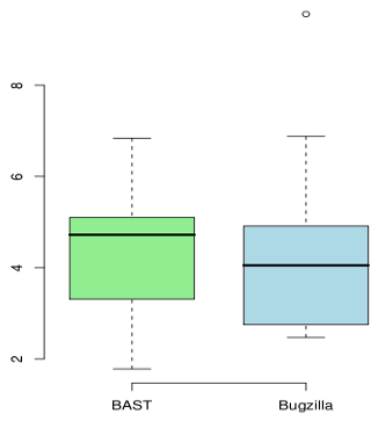

Figure 2. Time spent.

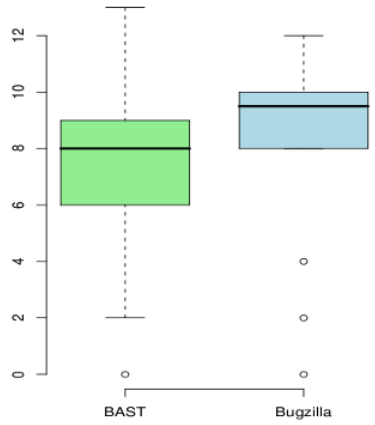

Figure 3. Dupls. avoided.

For duplicate bug reports avoided, BAST had mean value of 7.56 bug reports avoided and SD of 3.5, while Bugzilla had mean value of 8.33 and SD of 3.2. Once again, we need to plot boxes to understand the differences among these values, see Figure 3. From Figure 3, we can conclude that Bugzilla helped to avoid more duplicates than BAST. Although Bugzilla is a little better than BAST concerning this aspect, we can observe that more subjects using BAST can find higher number of duplicates than Bugzilla.

The descriptive analysis showed that Bugzilla had better performance than BAST in both cases: time spent on analysis and duplicates avoided. However, it is important to mention that the highest number of duplicate bug reports avoided were achieved with BAST and the minimum time spent on analysis too (see Table 2).

	Time spent on analysis		Bug-reports avoided	
	BAST	Bugzilla	BAST	Bugzilla
Mean	4.54	4.32	7.56	8.33
Maximum	6.84	9.56	13	12
Minimum	1.78	2.47	0	0
SD	1.49	1.91	3.5	3.2

Table 2. Descriptive statistics.

	Time spent on analysis	Duplicates avoided
t_0	0.6292	-1.2466
Degrees of freedom	17	17
p-value	0.5376	0.2294
T distribution	2.11	2.11
Result (t_0 ¿ T)	H_0: not rejected	H_0: not rejected

Table 3. t-tests. 95% of confidence.

T-test. The data collected during the experiment were submitted to the *t-test* with 95% of confidence. Table 3 summarizes the results of the test. In the analysis of the time saved with both tools, the *t-test* did not reject the null hypothesis. Thus, we can conclude that there was no gain using BAST instead of Bugzilla to save time during the analysis of bug reports. In the analysis of duplicates avoided, the *t-test* also did not reject the null hypothesis, concluding that there is no advantage in using BAST to avoid duplicates.

Analysis of dependency. Table 4 shows a matrix of correlations for the aspects of subjects profile. The correlations can variate from -1 to $+1$, and 0 (zero) means no correlation among the variables. As we can observe, there are no correlations among the characteristics of subjects profile and the time spent to analyze bug reports and amount of duplicates avoided.

	BAST time	Bugzilla time	BAST duplicates avoided	Bugzilla duplicates avoided
Years of expereince	-0.132974628	-0.02296878	-0.19721612	0.183007854
Number of projects	-0.113791496	0.3720158	-0.28693026	-0.020936575
Bug-trackers used	-0.167706238	0.3511591	-0.26207382	0.052027136

Table 4. Matrix of correlation.

Qualitative analysis. The result of the qualitative analysis about the BAST usability and functionality is summarized as follows.

BAST features. Seven (7) subjects used the *filter features* provided by the tool, while eleven (11) did not use it. From those that used the filters, all of them told that it was useful for the analysis of bug reports. Furthermore, three (3) subjects told that would be interesting if the tool also provided other types of filters. Seven (7) subjects told that the *ordering features* are useful, while eleven (11) participants did not use them.

BAST Usability. From the seven (7) participants that used the filters, only one mentioned some difficult to use it, and only one subject from those that used the ordering features had problem with it. Four (4) subjects experienced problems with the visualization of bug reports details. These problems appeared due to issues from the infra-structure.

BAST usefulness. Fifteen (15) subjects believe the way bug report details are presented in BAST is more useful for the analysis than Bugzilla.

Analysis Conclusion. The descriptive statistics showed that Bugzilla had a little better performance for both aspects studied: time spent on analysis and amount of duplicate bug reports avoided. However, the differences among the data analyzed for both tools in the descriptive statistics were not significant, which turns hard to draw a concrete conclusion saying what tool is better.

Furthermore, the t-test applied to test the hypotheses did not provide sufficient data to reject the null hypothesis. However, the qualitative analysis showed that BAST have many good aspects that should be taken into account before choosing one of the tools. It was clear that subjects felt more comfortable while using BAST than Bugzilla

due to its usability.

7. Research Contribution

This work brings some important contributions to the research and industry community, as it is described next: a *state-of-the-art* of the area; a *characterization study* of the problem; a *tool called BAST* to help mitigating the problem; two evaluations of the tool: a *case study* executed in an industry context and an *experiment* with students; and *final commercial product*. In addition to the contribution mentioned, some papers were produced and published:

- CAVALCANTI, Y. C.; ALMEIDA, E. S.; CUNHA, C. E. A.; LUCRÉDIO, D.; MEIRA, S. R. L. An Initial Study on the Bug Report Duplication Problem. In: 14th European Conference on Software Maintenance and Reengineering, Madrid, Spain. CSMR, 2010.
- CAVALCANTI, Y. C.; CUNHA, C. E. A.; ALMEIDA, E. S.; MEIRA, S. R. L. BAST - A Tool for Bug Report Analysis and Search. In: XXIII Simpósio Brasileiro de Engenharia de Software, Fortaleza, Brazil. XXIII SBES, 2009 (*awarded tool*).
- CAVALCANTI, Y. C.; MARTINS, A. C.; ALMEIDA, E. S.; MEIRA, S. R. L. Evitando Relatos de CRs Duplicadas em Projetos Open Source de Software. In: 9^o Fórum Internacional de Software Livre, Porto Alegre, Brazil. FISL, 2008 (*in portuguese*).
- CAVALCANTI, Y. C.; ALMEIDA, E. S.; CUNHA, C. E. A.; LUCRÉDIO, D.; MEIRA, S. R. The Bug Report Duplication Problem: A Characterization Study. Journal of Software Maintenance and Evolution: Research and Practice (*under review*).

In addition, this dissertation was published in format of book, titled "*A Bug Report Analysis and Search Tool*", by the LAP LAMBERT Academic Publishing press.

8. Concluding Remarks and Future Work

This work proposed and evaluated a solution to the bug reports duplication problem. As it was described, this problem is present in all the projects investigated, and the problem is characterized by the submission of two or more bug reports that describe the same software change/issue. The main consequence of this problem is the overhead of rework when managing these bug reports. In other words, people involved with bug report analysis, inevitably, spend time with search and analysis of existing bug reports, to ensure that duplicates will not be submitted.

In this sense, this work performed a characterization study of the problem, described a brief survey about the research area on mining bug repositories, described a tool developed to prevent duplicates submission, and performed also two evaluations of it: one involving a private organization and other with students. Since in this work it was developed and evaluated an initial prototype, we are aware that enhancements and features must be implemented, also as some defects must be fixed, and perform other evaluations.

Some important aspects for future work are described as follows: *evolve from prototype*, where a new version of tool should be released with recommended features and bug fixes; include *information visualization* techniques[11], in order to help users to perform the analysis of bug reports; add *alternative integration methods*, so BAST can be ease integrated with other tool not tested in this work; and, finally, improve *search and raking techniques*, such as putting tags and query reformulation [5] or providing other ranking techniques to improve search results precision.

Concerning the evaluations performed, *experiment replications* should be conducted. Although the qualitative analysis indicates that BAST is preferable to perform analysis and searches of bug reports, the quantitative analysis must be more conclusive, and this is possible by replicating our experiments.

Acknowledgement

This work was partially supported by the National Institute of Science and Technology for Software Engineering[3], funded by CNPq and FACEPE, grants 573964/2008-4, APQ-1037-1.03/08 and RHAE-559839/2009-0.

References

[1] Antoniol, G., Penta, M. D., Gall, H., and Pinzger, M. (2005). Towards the integration of versioning systems, bug reports and source code meta-models. *Electronic Notes in Theoretical Computer Science*, **127**(3), 87–99.

[2] Anvik, J. and Murphy, G. C. (2007). Determining implementation expertise from bug reports. In *Proc. of the 4th Inter. Work. on Mining Soft. Repositories (MSR'07)*. IEEE.

[3] Anvik, J., Hiew, L., and Murphy, G. C. (2005). Coping with an open bug repository. In *Proc. of the 2005 OOPSLA Work. on Eclipse Technology eXchange*, pages 35–39.

[4] Anvik, J., Hiew, L., and Murphy, G. C. (2006). Who should fix this bug? In *Proc. of the 28th Inter. Conf. on Soft. Eng. (ICSE'06)*, pages 361–370.

[5] Baeza-Yates, R. A. and Ribeiro-Neto, B. A. (1999). *Modern Information Retrieval*. Addison-Wesley.

[6] Basili, V., Selby, R., and Hutchens, D. (1986). Experimentation in software engineering. *IEEE Trans. on Soft. Eng.*, **12**(7), 733–743.

[7] Bennett, K. H. and Rajlich, V. T. (2000). Software maintenance and evolution: a roadmap. In *Proc. of the Conf. on The Future of Soft. Eng. (ICSE'00)*, pages 73–87.

[8] Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., and Zimmermann, T. (2007). Quality of bug reports in eclipse. In *Proc. of the 2007 OOPSLA Work. on Eclipse Technology eXchange (Eclipse '07)*, pages 21–25. ACM.

[9] Canfora, G. and Cerulo, L. (2005). Impact analysis by mining software and change request repositories. In *Proc. of the 11th IEEE Inter. Soft. Metrics Symposium (METRICS'05)*, page 29.

[10] Canfora, G. and Cerulo, L. (2006). Supporting change request assignment in open source development. In *Proc. of the 2006 ACM Symposium on Applied Comp. (SAC'06)*, pages 1767–1772. ACM.

[11] Card, S. K., Mackinlay, J. D., and Shneiderman, B. (1999). *Readings in Information Visualization: Using Vision to Think*. The Morgan Kaufmann Series in Interactive Technologies. Morgan Kaufmann.

[12] Eastwood, A. (1993). Firm fires shots at legacy systems. *Comp. Canada*, **19**(2), 17.

[13] Erlikh, L. (2000). Leveraging legacy system dollars for e-business. *IT Professional*, **2**(3), 17–23.

[14] Fischer, M., Pinzger, M., and Gall, H. (2003a). Analyzing and relating bug report data for feature tracking. In *Proc. of the 10th Working Conf. on Reverse Eng. (WCRE'03)*, pages 90–99.

[3]INES - `http://www.ines.org.br`

[15] Fischer, M., Pinzger, M., and Gall, H. (2003b). Populating a release history database from version control and bug tracking systems. In *Proc. of the 19th Inter. Conf. on Soft. Maintenance (ICSM'03)*, pages 23–32. IEEE.

[16] Hiew, L. (2006). *Assisted Detection of Duplicate Bug Reports*. Master's thesis, The University of British Columbia.

[17] Huff, F. (1990). Information systems maintenance. *The Business Quarterly*, (55), 30–32.

[18] Kagdi, H., Maletic, J., and Sharif, B. (2007a). Mining software repositories for traceability links. In *In the Proc. of the 15 IEEE Inter. Conf. on Program Comprehension (ICPC'07)*, pages 145–154.

[19] Kagdi, H., Collard, M. L., and Maletic, J. I. (2007b). A survey and taxonomy of approaches for mining software repositories in the context of software evolution: Survey articles. *Journal of Soft. Maint. and Evol.*, **19**(2), 77–131.

[20] Ko, A. J., Myers, B. A., and Chau, D. H. (2006). A linguistic analysis of how people describe software problems. In *Proc. of the Visual Languages and Human-Centric Computing (VLHCC'06)*, pages 127–134.

[21] Koponen, T. and Lintula, H. (2006). Are the changes induced by the defect reports in the open source software maintenance? In H. R. Arabnia and H. Reza, editors, *Proc. of the 2006 Inter. Conf. on Soft. Eng. Research (SERP'06)*, pages 429–435. CSREA.

[22] Lancaster, F. W. (1986). *Vocabulary Control for Information Retrieval*. Information Resources Press, 2 edition.

[23] Moad, J. (1990). Maintaining the competitive edge. *Datamation*, **4**(36), 61–62.

[24] Panjer, L. D. (2007). Predicting eclipse bug lifetimes. In *Proc. of the Fourth Inter. Workshop on Mining Soft. Repositories (MSR'07)*, page 29.

[25] Podgurski, A., Leon, D., Francis, P., Masri, W., Minch, M., Sun, J., and Wang, B. (2003). Automated support for classifying software failure reports. In *Proc. of the 25th Inter. Conf. on Soft. Eng. (ICSE'03)*, pages 465–475.

[26] Runeson, P., Alexandersson, M., and Nyholm, O. (2007). Detection of duplicate defect reports using natural language processing. In *Proc. of the 29th Inter. Conf. on Soft. Eng. (ICSE'07)*, pages 499–510. IEEE.

[27] Salton, G., Wong, A., and Yang, C. S. (1975). A vector space model for automatic indexing. *Commun. ACM*, **18**(11), 613–620.

[28] Sandusky, R. J., Gasser, L., and Ripoche, G. (2004). Bug report networks: Varieties, strategies, and impacts in a f/oss development community. In *Proc. of the 1st Inter. Work. on Mining Soft. Repositories (MSR'04)*.

[29] Sommerville, I. (2007). *Software Engineering*. Addison Wesley, 8 edition.

[30] Song, Q., Shepperd, M. J., Cartwright, M., and Mair, C. (2006). Software defect association mining and defect correction effort prediction. *IEEE Trans. on Soft. Eng.*, **32**(2), 69–82.

[31] Wang, X., Zhang, L., Xie, T., Anvik, J., and Sun, J. (2008). An approach to detecting duplicate bug reports using natural language and execution information. In *Proc. of the 13th Inter. Conf. on Soft. Eng. (ICSE'08)*, pages 461–470. ACM.

[32] Weiss, C., Premraj, R., Zimmermann, T., and Zeller, A. (2007). How long will it take to fix this bug? In *Proc. of the 4th Inter. Work. on Mining Soft. Repositories (MSR'07)*, pages 20–26. IEEE.

[33] Wohlin, C., Runeson, P., Martin Höst, M. C. O., Regnell, B., and Wesslén, A. (2000). *Experimentation in Software Engineering: An Introduction*. The Kluwer Inter. Series in Soft. Eng. Kluwer Academic Publishers.