One Step More to Understand the Bug Report Duplication Problem

Yguaratã Cerqueira Cavalcanti[1,5], Paulo Anselmo da Mota Silveira Neto[1,5],
Eduardo Santana de Almeida[2,5], Daniel Lucrédio[4,5],
Carlos Eduardo Albuquerque da Cunha[1], Silvio Romero de Lemos Meira[1,3,5]

[1]Center for Informatics – Federal University of Pernambuco (UFPE)
[2]Computer Science Department – Federal University of Bahia (UFBA)
[3]Recife Center for Advanced Studies and Systems (C.E.S.A.R.)
[4]Computing Department – Federal University of São Carlos (UFSCar)
[5]Reuse in Software Engineering (RiSE)

{ycc,pamsn,ceac,srlm}@cin.ufpe.br, esa@dcc.ufba.br, daniel@dc.ufscar.br

Abstract

According to recent work, duplicate bug reports impact negatively on software maintenance and evolution productivity due to, among other factors, the increased time spent on report analysis and validation. Therefore, a considerable amount of time is lost mainly with duplicate bug report analysis. In this sense, this work presents am exploratory study using data from bug trackers from private and open source projects, in order to understand the possible factors (i.e. software life-time, size, amount of bug reports, etc.) that cause bug report duplication and its impact on software development. This work also discusses bug report characteristics that could help identifying duplicates.

1 INTRODUCTION

With the objective of improving the software maintenance and evolution, some organizations use specific systems to manage, track and store change requests. These systems are generally called *bug report tracking systems*, *bug repositories*, or just *bug trackers* [1, 2]. 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 by developers, users and testers. Such systems are useful because software changes can be identified and reported to developers quickly [3], and bug report historical databases can be used as documentation for the project history.

Although bug trackers bring many benefits to software development, some challenges may appear as a consequence of its usage, such as: dynamic assignment of bug reports [4, 5, 6, 7], quality of bug report descriptions [8, 9, 10, 11], and duplicate bug reports detection [3, 12, 13, 14, 15, 16, 17]. Furthermore, there are possibilities to improve software development using the bug tracker historical database, such as: change impact analysis and effort estimation [18, 19, 20] and software evolution and traceability [21, 22, 23, 24].

The focus of this work is on the *bug report duplication problem*. This problem is characterized by the submission of two or more bug reports that describe the same change request. The main consequence of this problem is the rework when managing these bug reports. For example, the staff responsible for the search and analysis of bug reports must spend a considerable amount of time with bug reports that have already been submitted. To perceive the critical aspect of this problem, recent work [3, 14] have shown that between 10% to 30% of a bug report repository is composed of duplicate bug reports. However, it is important to highlight that we are not considering the scenario when two bug reports describe different issues, but when deeply investigated, there is a unique root case for the issues.

Additionally, in a previous work [25] we described an initial study that addressed some questions about the factors that could cause bug report duplication and its impact on software development. Such work analyzed several open source projects and one private project. The questions and the answers achieved are described next:

Do the analyzed projects have a considerable amount of duplicate bug reports? It was observed that every project had many duplicates in their repositories, varying from 8% to 60% of the total repository size, which confirmed some preliminary results [3, 14];

978-0-7695-4273-7/10 $26.00 © 2010 IEEE
DOI 10.1109/SBES.2010.12

Is the submitters productivity being affected by the bug report duplication problem? The work concluded that a lot of time is wasted just with search and analysis of bug reports. Such activities are intensively performed because of the huge presence of duplicate bug reports;

How are the relationships between master bug reports (the first entry of a set of related reports) and duplicate bug reports characterized? Answering this question is useful to drive a possible solution for problem. It was observed that most of the relationships between master and duplicates is characterized by the one-to-one relationship. It means that one master bug report is associated with only one duplicate. In such scenario, it can be concluded that techniques such as machine learning – which have been approached in [3] – is not applicable to the problem;

Does the type of bug report influence the amount of duplicates? Basically, there are two types of bug reports: those that describe enhancements and those that describe defects. It was concluded that the probability of duplicates occurrence is independent of such types;

Is the submitter profile an important factor that could impact on the bug report duplication problem? It was found that the profile of the submitter is an important factor for the duplicates submission. For example, if a person submitting a new bug report is a beginner in the project, then chances are that such report is duplicate.

In this current work, we extended the previous one with new questions to better understand the duplication problem. Thus, this work presents an exploratory study about search and analysis of bug reports, in order to understand other factors for duplication and opportunities to improve avoiding it. It was used both the private and open source projects from the initial study. Additionally, the new results are used to provide useful advice to avoid duplicates, along with a description of how duplicate bug reports could impact the entire life-cycle of the software development process.

The remainder of this paper is structured as follows: Section 2 presents the definition of the study; Section 3 describes the projects and data used in the study. In Section 4, the operation is discussed, and in Section 5 the analysis and interpretation of the results are presented. Sections 6, 7, 8 and 9 present the threats to validity, final considerations, related work and conclusion, respectively.

2 DEFINITION OF THE STUDY

The GQM method [26] was used to define this characterization study. The GQM consists of the definition of the study goal, the questions to be answered, and the related metrics that must be collected to help answering the questions. For all the metrics presented in this current work, we did not find values in related work to serve as a baseline.

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*. The following questions and metrics were defined.

Question 1. *Is the software size a factor for duplication?* The metric for this question is related to the number of LOC (Lines Of Code) that a software project had until the data selection. The LOC, in this study, does not count comments and blank lines. The values for this variable were obtained through the site `http://www.ohloh.net`. We believe that the number of LOC can influence the amount of errors, and consequently more bug reports would be submitted, increasing the chances of duplication;

Question 2. *Is the software life-time a factor for duplication?* This question's metric is related to the time (in years) a software project has been in development. We computed the life-time of a software from the first bug report submission. Anvik et. al. [3] analyzed projects with 7 and 9 years, however, we must analyze projects with different life-times to understand if it is a factor for bug report duplication;

Question 3. *Is the bug repository size a factor for duplication?* Our metric in this question is related to the amount of bug reports that a project has in its bug repository. With this metric, we can analyze whether large bug repositories are more susceptible to the submission of duplicates or not. In [3], the bug repositories had a considerable amount of bug reports, however, we wanted to understand if duplicates are present in small bug repositories too;

Question 4. *Is the number of submitters a factor for bug report duplication?* The metric is related to the amount of submitters in the period that we collected the bug reports. We consider the number of submitters as being equal to the number of people who submitted bug reports in the period. Also in [3], the analyzed projects had many submitters, but we would like to understand if duplicates are also present in projects with few a submitters;

Question 5. *Is the amount of staff a factor for bug report duplication?* The metric is related to the number of people involved in the project development. We consider the number of developers as being equal to the number of people assigned to resolve a bug report. Related work [3] showed that projects with a large staff have many duplicates, thus we would like to understand if this is also true for projects with a small staff;

Question 6. *Is the submitter's profile a factor for duplication?* The metric is related to the type of submitter profile, and we want to understand what type is more susceptible to submitting duplicates. The values for this metric are: *sporadic* (**S**), *average* (**A**) and *frequent* (**F**). *Sporadic* is the reporter who submitted at most 10 bug reports in the

analyzed period, *average* is the person who submitted between 10 and 30 bug reports in the period, and *frequent* is the person who submited more than 30 bug reports in the period. We defined these threshold values according to previous analysis of the repositories used in this work;

Question 7. *Does the vocabulary used in bug report descriptions influence on duplication problem?* It is believed that a controlled vocabulary could help avoiding duplicate bug reports [27]. For example, with a well defined vocabulary, the submitters could perform better searches using keywords closer to those present in the new bug report. Thus we want understand if projects with more concise vocabulary has less duplicates than others. The consistency of the vocabulary is analyzed through the use of common words in bug report descriptions. Thus, the variable is `common_words`, and this measure does not count *stop words*[1] of the English language. Moreover, we analyzed only summary and long description fields of the bug reports.

Question 8. *Is the amount of comments in bug reports an indicator of possible duplicates to be submitted?* The hypothesis is that the number of comments of a bug report can be a potential indicator that it is likely to receive duplicates. To investigate this issue, we need to do the following: given that a bug report has duplicate entries, we must investigate if this bug report contains more comments than usual, when compared to those that do not have duplicates. Thus, the variable is `number_of_comments` and this measure counts the comments of bug reports.

3 PROJECTS AND DATA SELECTION

To conduct this study, we used the same projects from our initial work [25]: eight (8) open source projects and one (1) private project developed at C.E.S.A.R[2]. Regarding the open source projects, all bug reports until the end of June/2008 were used. For the private project, all bug reports were from November/2006 to March/2008, which includes the entire life-cycle for this project. Table 1 summarizes the projects and its characteristics.

Project	Domain	Code size	Staff size	Bugs	Life-time
Bugzilla	Bug tracker	55K	340	12829	14
Eclipse	IDE	6.5M	352	130095	7
Epiphany	Browser	100K	19	10683	6
Evolution	E-mail client	1M	156	72646	11
Firefox	Browser	80K	514	60233	9
GCC	Compiler	4.2M	285	35797	9
Thunderbird	E-mail client	310K	192	19204	8
Tomcat	Application server	200K	57	8293	8
Private Project	Mobile application	2M	21	7955	2

Table 1. Projects characteristics.

[1] *Stop words* are words that do not improve the searches.

[2] Recife Center for Advanced Studies and Systems.

Bugzilla. It is a software for bug report tracking. This projects is hosted by Mozilla Foundation.

Eclipse. It is an open development platform with support to C, PHP, Java, Python and other languages.

Epiphany. It is the official web browser for, and hosted by, the Gnome desktop.

Evolution. It is a desktop e-mail client. It provides integrated e-mail, address book and calendar features to the users of the Gnome desktop. It is also hosted by Gnome.

Firefox. It is a popular web browser that runs in a variety of platforms (Windows, Linux, Mac etc). Firefox is also hosted by Mozilla.

GCC. The GCC is a collection of compilers, including front ends for C, C++, Objective C, Fortran, Ada and Java.

Thunderbird. It is a cross-platform desktop email client hosted and developed by Mozilla Foundation.

Tomcat. Apache Tomcat is an implementation of the Java Servlet and JavaServer Pages technologies, developed by the Apache Foundation.

Private Project. In this project, applications for mobile devices are developed and tested. We chose this project because it was the biggest project that we had fully access. Until this work, there were 60 testers divided into 6 teams.

4 STUDY EXECUTION

This study was performed at C.E.S.A.R., from June/2008 to August/2008, and at UFBA (Federal University of Bahia) from March 2010 to May 2010. Python scripts were built to read the bug reports in XML format from each project. These XMLs are detailed later. Using these scripts we were able to obtain the values for the metrics.

The bug reports from each project were exported from their bug trackers in XML format. This was possible because the bug trackers used in these projects provide such functionality. The following listing shows an example of a bug report exported in XML format from Bugzilla. Some fields were removed for simplicity.

```
<bugzilla>
  <bug>
    <bug_id>391316</bug_id>
    <creation_ts>2007-08-07...</creation_ts>
    <short_desc>Add FTP support</short_desc>
    <delta_ts>2007-08-07...</delta_ts>
    ...
    <reporter>tt@mozilla.com</reporter>
    <bug_when>2007-08-07 20:23:16</bug_when>
  </bug>
</bugzilla>
```

The material used in this study can be accessed through the site `http://www.cin.ufpe.br/~ycc/bug-analysis`. This URL also contains the set of bug

reports used in this study, as well as the queries used to extract them from the repositories. We did not publish the private project's data due to confidentially reasons.

5 ANALYSIS AND INTERPRETATION

This section analyzes the questions defined in Section 2, according to the metrics obtained from each project, and presents a descriptive statistics analysis. At the end of this section, we also present a statistical correlations analysis to confirm the former and the main findings.

Question 1. Is the software size a factor for duplication? According to the graph of Figure 1, there is no established pattern that leads us to conclude that there is some relationship between the number of LOC and the number of duplicates of a project. For example, the Eclipse project has the greatest number of LOC but it is the third project with less duplicate bug reports. In another example, the Tomcat project has a number of LOC similar to the Thunderbird project, but the difference in the number of duplicates is about 40%.

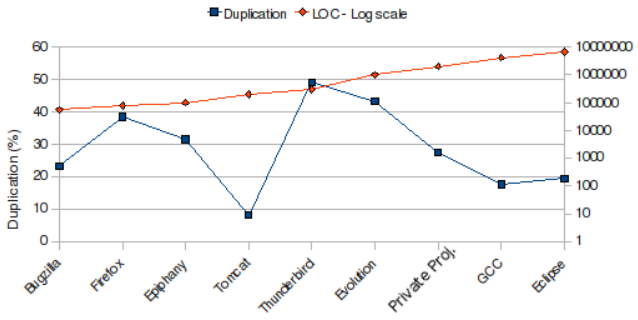

Figure 1. Duplication and LOC.

Question 2. Is the software life-time a factor for duplication? As shown in the chart of Figure 2, we observed in this study that the life-time is not a factor for the duplication problem. Projects with higher life-time do not necessarily have more duplicate bug reports than projects with smaller life-time. Thunderbird and Tomcat projects are good examples of this observation; both have similar life-times, but differ sharply on the amount of duplicates. Another example is Bugzilla and the private project: the first is 12 years older than the second, however it has about 10% less duplicates.

Question 3. Is the repository size a factor for duplication? The amount of bug reports in repositories is not a factor causing the problem of duplication, as shown in the chart of Figure 3. This finding contradicts our initial thoughts. We believed that from the moment that the bug report database increases, it would be more difficult to find similar bug reports. However, we must take into account

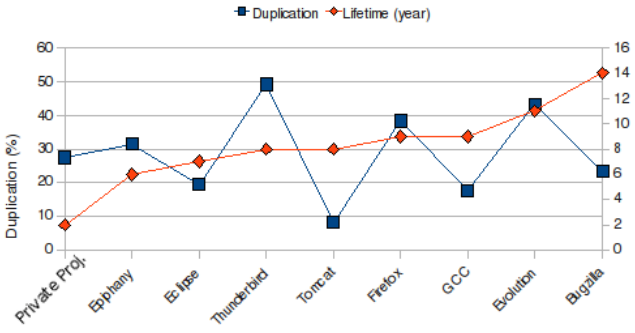

Figure 2. Duplication and life-time.

that as the amount of bug reports increases, the knowledge of the submitters on the repository also increases, which balances the repository growth factor.

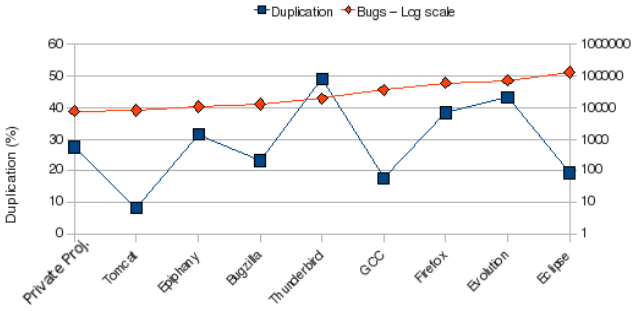

Figure 3. Duplication and repository size.

Question 4. Is the number of submitters a factor for bug report duplication? Figure 4 shows the relationship between the quantity of duplicates, number of submitters, and staff size. Each chart shows the situation of a specific project. In some graphics, the time does not match the total life-time of the project, because we considered, in that case, the year when the first duplicate bug reports were submitted.

Most projects have a similar tendency in their graph curves regarding the number of submitters and duplicates: the number of duplicates tends to increase together with the number of submitters, but it tends to decrease faster. One explanation for this fact is that people involved in projects get a more uniform vision of the bug repository as time passes, with greater knowledge of bug reports that have already been submitted. Firefox and Epiphany projects exemplify this fall in the number of duplicates without a significant decrease in the amount of submitters.

Another interesting observation is that there always seems to be a considerable increase in the number of duplicates in the first year of the projects. This could be because the first year of the project often coincides with the first release available to users and testers, which is when they start to submit bug reports. For example, the Bugzilla

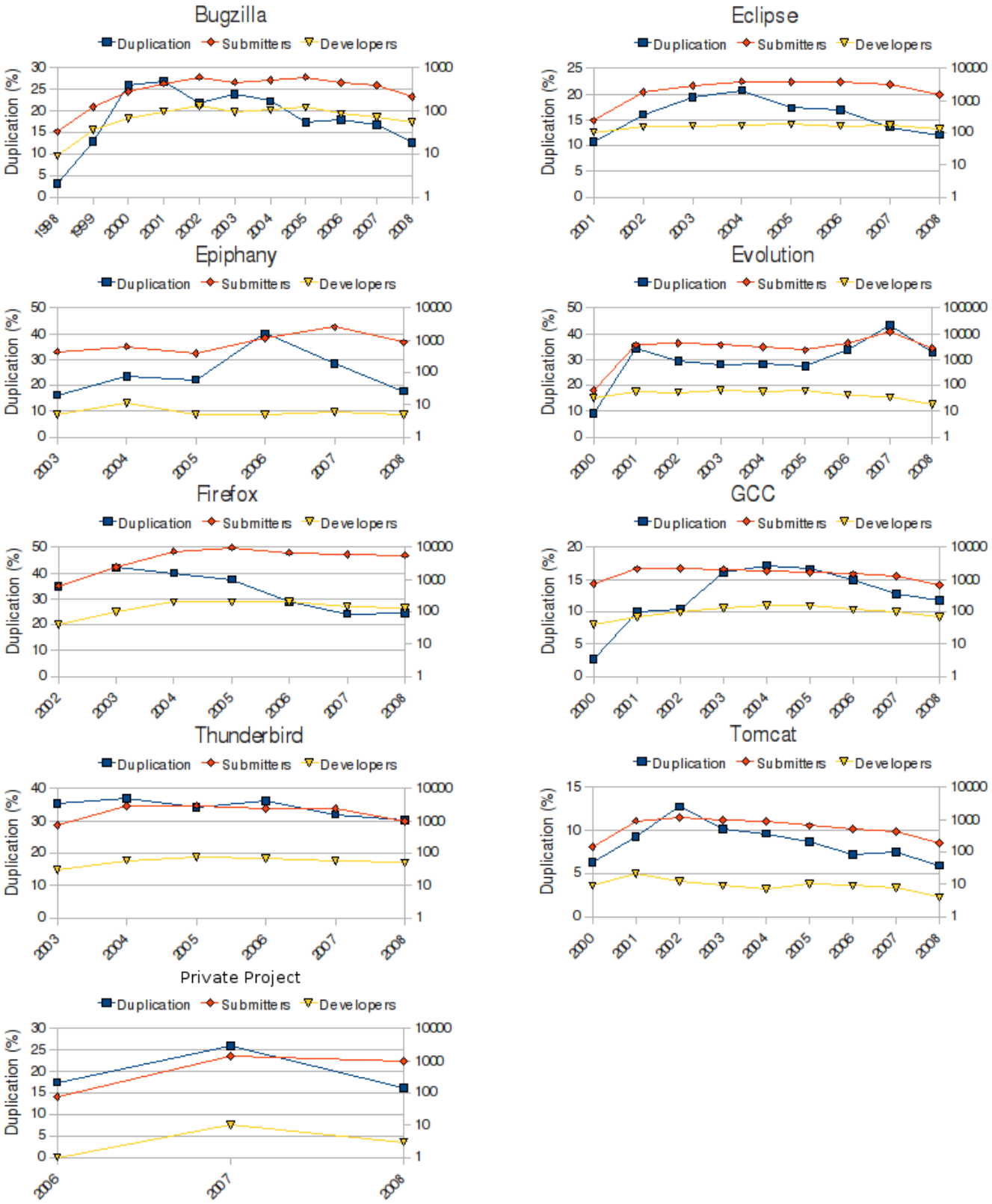

Figure 4. Duplication, staff size, and submitters. Data reduced using Log10 for better visualization.

project has 14 years of life-time, but the first duplicate bug report appeared only in 1998, when it launched the first release available to users (version 2.0). This also seems to be true for major changes and new releases, like years 2006 and 2007 for the Evolution project.

Question 5. Is the amount of staff a factor for bug report duplication? According to Figure 4, the staff size has not suffered considerable changes in almost all projects, so it is not possible to establish a direct relationship with the amount of duplicates. In contrast, the number of developers seems to be related with the number of submitters. After that finding, we calculated that an average of 76%[3] (per project) of the developers are also submitters. From these 76%, an average of 13 submitters are among the 20 most active ones. So, when there is a drop in the number of developers, there is also a drop in the amount of submitters.

Question 6: Does the vocabulary used in bug report descriptions influence on the duplication problem? We analyzed the descriptions of the bug reports from Eclipse, Firefox and the private project. We were not able to examine the others because their duplicate bug reports did not specify the master bug reports, thus not enabling us to complete the bug report groupings. The values for the 3 projects are considered very close, as shown in Figure 5. We do not have previous data for this metric to determine if these values are low or high, however we believe that more words should be shared between duplicate bug reports in order to facilitate the identification.

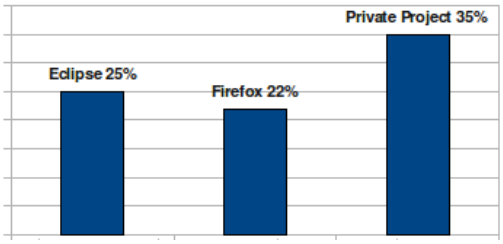

Figure 5. M_5: percentage of common words.

Another interesting finding is the fact that the private project's bug reports share more common words than the other ones. We believe this happens because the environment in the private project is more restricted and controlled due to the reduced number of staff members and submitters and the use of templates to describe the issues, while in distributed open source projects any user can submit a bug report. However, more studies should be performed with private projects to confirm such explanation.

Question 7. Is the submitter's Profile a factor for duplication? Figure 6 shows the contribution of each type of submitter to the number of duplicates of each project. As it can be seen, most of the duplicate bug reports are submitted by sporadic submitters, followed by average and frequent submitters. The average and frequent submitters contribute very little to the problem of duplication, possibly because these are people who are longest in the project and had good knowledge on the repository.

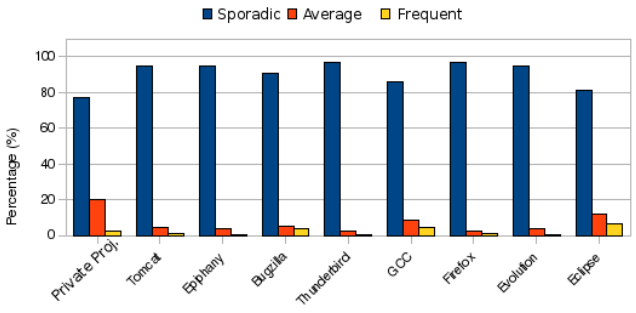

Figure 6. Duplication and submitter profile.

We observed that sporadic submitters are more prone to submit duplicate bug reports. One possible explanation is that they probably have no sufficient knowledge about the project and its bug repository, or they are not aware of the duplication problem. Often these people also have no experience with bug report tracking systems, where the searches must be carried out. Moreover, according to our analysis, most of sporadic submitters submitted at most 1 or 2 bug reports throughout the life-time of the project.

Question 8. Is the amount of comments in bug reports an indicator of possible duplicates to be submitted? Before analyzing such question, first we prepared the data as described by [17]. Such preparation consists of unifying the related islands of duplicate bug reports. Such islands exist because, as the projects' repositories grow, the developers are no longer able to identify the master bug report for an incoming duplicate, thus selecting any other bug report that is also related to that duplicate to be the master. After the unification of such islands, each master bug report is related to only one group of duplicates, which increases the precision of our comment analysis.

We analyzed the amount of comments in master bug reports from the duplicate groups, and the amount of comments of the bug reports that did not receive duplicates. In average, master bug reports have 4 to 5 times more comments than bug reports without related duplicates. This is an indication that frequently commented bug reports are more likely to receive duplicates than others. In this sense, it is reasonable to include the number of comments as a parameter of the algorithm used to rank the result of bug report searches. Thus, bug reports with more comments will be more relevant than bug reports with less comments, increasing the chances of identifying a duplicate.

[3]We calculated this value by observing the submitters that are also present in the source code commits. We repeated this observation for each project.

Correlation Analysis. In previous paragraphs we used only descriptive statistics to analyze the influence of projects characteristics on duplication problem. In this section, we provide an analysis of correlation that was carried out in order to identify if the characteristics of the software projects are statistically related. We used the *Pearson* correlation to perform the calculations. The values for the correlation analysis can vary from -1 to +1, where as much as closer to -1 means that there is no correlation, while values closer to +1 mean a strong correlation.

Table 2 summarizes such analysis. As it can be seen, the analysis using statistical correlations has confirmed our initial descriptive analysis.

Correlation	*p-value*	Intensity
LOC *vs.* Duplication	0.35	Weak/Moderate
Amount of staff *vs.* Duplication	0.09	Weak
Amount of bugs *vs.* Duplication	0.04	Weak
Life-time *vs.* Duplication	0.06	Weak
Amount of submitters *vs.* Duplication	0.51	Moderate
Sporadic submitters *vs.* Duplication	0.84	Strong
Average submitter *vs.* Duplication	0.60	Moderate
Frequent submitter *vs.* Duplication	0.43	Weak
Amount of comments *vs.* Duplication	0.80	Strong

Table 2. Analysis of correlations.

Main Findings of the Study. This study was conducted with the goal of understanding which characteristics of the projects can be factors for bug report duplication. The main findings of this study are:

MF1. The number of LOC is a weak/moderate factor for the duplication problem;

MF2. The size of the repository does not seem to be a factor for duplication;

MF3. The projects' life-time does not seem to be a factor for duplication;

MF4. The amount of staff seems to be a moderate factor for the duplication problem;

MF5. The amount of submitters does not seem to be a determining factor for duplication problem;

MF6. 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;

MF7. 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 an explicitly defined vocabulary to control bug report descriptions;

MF8. We noticed that submitters from private project are more likely to submit reports using a common vocabulary than submitters from open source projects; and

MF9. Bug reports with a considerable amount of comments are also the bug reports more prone to receive duplicate bug reports.

6 THREATS TO VALIDITY

Envisioning a possible replication of this study and the generalization of the results, we have identified the following threats to the study's validity.

Bug trackers. In our study, we analyzed bug reports from two different bug trackers, but one of them is an internal tool from the private project, without public access. Hence, projects using other types of bug trackers should be included in the analysis.

Statistical analysis mechanisms. It was only used descriptive statistics for this characterization study. However, we believe that this study is a starting point for others. New approaches should formalize and test hypotheses based on our results using, for example, probabilistic models and formal correlation techniques [28].

Negative results. We investigated some aspects for the projects that could be possible causes for the duplication issue, but most of them were not confirmed. We only identified a possible correlation with the submitters' profile and the number of submitters. Thus, more in-depth analysis for the negative results must be provided to increase the validity of the results.

Projects population. We analyzed 8 open source projects and 1 private project. The open source projects are quite representative of the population of that type of project, which allows us to generalize the results to the rest of the population with a good degree of confidence.

On the other hand, our analysis of just one private project does not allow us to generalize the results for that type of projects. However, we believe that private projects that have similar characteristics than what we analyzed (i.e.: LOC, submitters, if it is dedicated only to test software, etc) are also being similarly affected by the problem of duplicate bug reports.

Correlation vs. Causation. We must not confound correlation with causation. The possible correlations indentified in this study do not necessary mean a relationship of cause. The causes can be external factors that were not addressed in this work, thus further investigation must be performed.

7 FINAL CONSIDERATIONS

In the context of software maintenance and evolution processes, the impact of duplicate bug reports can be seen almost over all its activities. Generally, the maintenance and evolution process involves the activities of: identifying change requests; reporting these change requests (as bug reports); verifying the impact of these requests; planning the

implementation of change requests (i.e. which release will implement the changes); thus implementing such changes; and releasing the new software version with the changes [29].

In the first step of the process mentioned before, people responsible for identifying change requests must perform search and analysis of previous bug reports in order to not submit duplicate ones. However, if this search and analysis fails to avoid duplicate submissions, all the other following steps will be compromised by unnecessary rework. Thus, as a consequence of this inability of avoiding duplicates to be submitted in the first place, in practice people end up searching for duplicates in almost all steps of the process to ensure that rework will not be performed. This requires extra time and work. On the other hand, if the duplicates could be avoided in the first stage, all this time spent on search and analysis could be reused in other tasks.

As a workaround to this problem, some organizations have a team dedicated only to analyze the incoming bug reports, called CCB (Change Control Board). The CCB acts by analyzing all incoming bug reports to ensure that no invalid bug reports (such as duplicates) end up passing through the next steps of the software maintenance and evolution. However, even with the presence of a CCB, some duplicates still remain unidentified [3]. There are many reasons for that, such as: lack of attention or experience of the CCB, the amount of bug reports that are submitted per day; lack of good tools for the search and analysis of bug reports; or even the lack of knowledge over the repository.

The cost of a duplicate bug report that passes towards the entire software life-cycle, without being identified at the early steps, is very difficult to be measured. The cost of the time lost with duplicates is transferred to everyone involved directly or indirectly with the software being developed, which we can include testers, developers, managers, partners, co-related projects, etc.

7.1 Advice to Avoid Duplicates

The findings of this study and the previous one [25] can help to establish some actions in order to improve the current stage of the duplicates identification. So, next we list some advice that could be helpful:

Be sure to look at the submitter's profile before analyzing a bug report. As we could understand in the results of the study, people with a sporadic profile seem to be more prone to submitting duplicates. Thus, we suggest that some mechanisms are implemented to analyze the profile of the submitter, so the staff responsible for analyzing the incoming bug reports can decide whether to give more or less attention to a bug.

One way to achieve the prediction of a profile is looking at the amount of submitted bug reports by the person who is submitting a new one. Another way is to check when the submitter signed in into the bug repository. The bug repository administrator should decide which is better for their project.

Control the vocabulary and increase the granularity of fields in the bug report form. Since similar bug reports are described with different words, the identification of duplicates becomes difficult. So in that context, using a controlled vocabulary may be a way to avoid duplicates. This could be done by defining classes of words that a submitter could use to describe a bug report, as well as to divide the fields of a bug report so that more detailed information can be entered. For example, putting a specific field for execution information *traceback* [8, 16] would facilitates the use of techniques to find duplicate entries based on such type of information.

Improving the search algorithms. One of the best ways to reduce the number of duplicates is by improving the searching algorithms. We believe that it is the key point for future research in this field. In this context, we have also identified that bug reports with a considerable amount of comments are also the bug reports more prone to receiving duplicate bug reports. This is not difficult to understand, since the most commented bug reports are those that people are facing more frequently.

We have implemented a search engine [30] that uses the number of comments to influence the search results, as proposed before. During the initial tests, we observed that this really seems to improve the identification of duplicates. Moreover, this solution is simple to implement in the existing tools, since most of them provide a plug-in architecture, and thus this is an excellent opportunity for future work.

The launchpad[4] tool, which has an integrated bug tracker, has just implemented such feature to rank the search result of bug reports according to the number of comments. However, further empirical studies must be conducted to give more precise results about this technique.

In addition, we understand that reducing the number of duplicates does not reduce necessarily the time to search and analyse bug reports – which is the key point of the problem. However, this reduction is important in order to avoid rework in the other steps of software maintenance and evolution.

Improving the analysis method of bug reports. The most common way to perform analysis to avoid duplicates is to look at the reports one by one, while performing searches. This is clearly a hard and time consuming work, and not so efficient as we can see in the numbers shown in this wok. Thus, one way to improve duplicates detection is to improve the analysis process itself.

The Mozilla's projects have tried to improve the duplicates detection by showing the *all time top 100* bug reports,

[4] `https://launchpad.net/`

as a manual analysis alternative. In this way, before submitting a new issue, the submitter should look at this list – which is the list with most frequent submitted issues – to avoid submitting a duplicate.

Another way to improve bug report analysis is using computational techniques. For example, we believe that *information visualization* techniques [31] are a good promise to aiding this activity. One way to use such techniques would be mixing search algorithms and merging algorithms with graphical visualization, in a such a way that the submitter could visualize multiples similar bug reports at the same time and decide whether is a duplicate.

8 RELATED WORK

The first related work was from John Anvik [3]. He analyzed potential problems raised by bug repositories from Eclipse and Firefox projects, such as dynamic assignment of bug reports and bug report duplication. Although our work focused only in the duplication problem, we believe that it is a complement of Anvik's work, since we expanded the number of projects and variables analyzed in order to understand the problem.

Hiew's work [13] presented an approach to detect duplicate bug reports using clustering techniques. Runeson's work [14] addressed the bug report duplication problem using Natural Language Processing (NLP) [32]. While Wang et al. [16] proposed an approach to detect duplicate bug reports using NLP and execution information. The execution information consists of data about software execution when an error occurs, such as method calls or variable state.

Another work to detect duplicate bug reports was performed by Jalbert and Weimer [15]. They proposed a system to automatically classify duplicate bug reports as they arrive. As a differential from the others work, the system used surface features and textual semantics in conjunction with clustering techniques.

Bettenburg et al. [10, 17] tried a different approach. Instead of avoiding duplicates, they propose to merge the duplicate bug reports to assist developers with additional information (such as *tracebacks*, or even descriptions from different points of view).

The work of Sandusky et al. [33] differs substantially from the previous ones. It proposed a method to identify and visualize bug report networks. With such networks, it was possible to analyze relationships and dependencies among bug reports.

In our work we did not implement any technique to prevent or suggest duplicate bug reports. Instead, we studied several projects and their characteristics to try to understand the causes and effects involved in the submission of duplicate bug reports in the context of software development. We believe that empirical studies are an important fist step to understand the problem with a certain degree of confidence and define new directions for solutions.

9 CONCLUSION

In this work, we presented a complementary characterization study for a previous work [25] about the factors that may have impact on the bug report duplication problem. The study was performed using 9 bug report repositories from open source and private projects, both with different characteristics, such as software size, staff, life-time, number of bug reports, among others.

We identified which project characteristics may be influencing factors to the duplication problem and which does not seem to be related to it. For example: the number of LOC and life-time does not seem to have a direct influence on the problem, which contradicts our initial intuition; similarly, increased amounts of bug reports in the repository does not seem to be a factor that causes duplicate bug reports.

Furthermore, according to our main findings in the study, we presented some advice that could be useful to help avoiding duplicate bug reports, along with a description of how duplicate bug reports could impact on the entire lifecycle of the software development process. Indeed, we understand that it is not possible to reduce to zero the number of duplicates in a project, or to completely avoid the rework caused by it.

For future work and replications of this study, more private and open source projects should be included. It would be even better if the projects could be distributed into groups of related projects, sharing similar characteristics, thus regression analysis could be performed [34]. Some of the findings presented will be used as heuristics in two prototype applications that has been developed for search and analysis of bug reports [30, 35].

Acknowledgement. This work was partially supported by the National Institute of Science and Technology for Software Engineering[5], funded by CNPq and FACEPE, grants 573964/2008-4, APQ-1037-1.03/08 and RHAE-559839/2009-0.

References

[1] J. N. Johnson and P. F. Dubois. Issue tracking. *Comp. in Science and Eng.*, 5(6):71–77, November 2003.

[2] N. Serrano and I. Ciordia. Bugzilla, itracker, and other bug trackers. *IEEE Soft.*, 22(2):11–13, March-April 2005.

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

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

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

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

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

[7] G. Jeong, S. Kim, and T. Zimmermann. Improving bug triage with bug tossing graphs. In *Proc. of the 7th joint meeting of the European Soft. Eng. Conf. and the ACM SIGSOFT Symposium on the Foundations of Soft. Eng. (ESEC/FSE'2009)*, 2009.

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

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

[10] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim. Extracting structural information from bug reports. In *Proc. of the 2008 international workshop on Mining Soft. repositories (MSR'08)*, pages 27–30. ACM, 2008.

[11] M. Castro, M. Costa, and J.-P. Martin. Better bug reporting with better privacy. In *Proc. of the 13th Inter. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS XIII)*, pages 319–328. ACM, 2008.

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

[13] L. Hiew. Assisted detection of duplicate bug reports. Master's thesis, The University of British Columbia, 2006.

[14] P. Runeson, M. Alexandersson, and O. Nyholm. 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, May 2007.

[15] N. Jalbert and W. Weimer. Automated duplicate detection for bug tracking systems. In *The 38th Annual IEEE/IFIP Inter. Conf. on Dependable Systems and Networks (DSN'08)*, pages 52–61. IEEE, June 2008.

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

[17] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim. Duplicate bug reports considered harmful? In *Inter. Conf. on Soft. Maintenance (ICSM'08)*, pages 337–345. IEEE, 2008.

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

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

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

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

[22] M. Fischer, M. Pinzger, and H. Gall. 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, September 2003.

[23] M. D'Ambros and M. Lanza. Software bugs and evolution: A visual approach to uncover their relationship. In *Proc. of the 10th European Conf. on Soft. Maintenance and Reengineering (CSMR'06)*, pages 229–238. IEEE, March 2006.

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

[25] Y. C. Cavalcanti, E. S. de Almeida, C. E. A. da Cunha, D. Lucrédio, and S. R. de Lemos Meira. An initial study on the bug report duplication problem. In *Proc. of the 14th European Conf. on Soft. Maintenance and Reengineering (CSMR'2010)*, pages 273–276. IEEE, 2010.

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

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

[28] C. Wohlin, P. Runeson, M. C. O. Martin Höst, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: An Introduction*. The Kluwer Internation Series in Soft. Eng. Kluwer Academic Publishers, Norwell, Massachusets, USA, 2000.

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

[30] Y. C. Cavalcanti, C. E. A. da Cunha, E. S. de Almeida, and S. R. de Lemos Meira. Bast - a tool for bug report analysis and search. In *XXIII Simpósio Brasileiro de Engenharia de Software (SBES'2009)*, Fortaleza, Brazil, 2009.

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

[32] R. Feldman and J. Sanger. *The Text Mining Handbook: advanced approaches in analyzing unstructured data*. Cambridge University, 2007.

[33] R. J. Sandusky, L. Gasser, and G. Ripoche. Bug report networks: Varieties, strategies, and impacts in a f/oss development community. In *Proc. of the 1st Inter. Workshop on Mining Soft. Repositories (MSR'04)*, pages 80–84, May 2004.

[34] W. G. H. George E. P. Box, J. Stuart Hunter. *Statistics for Experiments*. Wiley, second edition, 2005.

[35] C. E. A. da Cunha, Y. C. Cavalcanti, P. A. da Mota Silveira Neto, E. S. de Almeida, and S. R. de Lemos Meira. A visual bug report analysis and search tool. In *Inter. Conf. on Soft. Eng. and Knowledge Eng. (SEKE'2010)*, 2010.