Software Qual J (2013) 21:39–66
DOI 10.1007/s11219-011-9164-5

The bug report duplication problem: an exploratory study

Yguaratã Cerqueira Cavalcanti · Paulo Anselmo da Mota Silveira Neto · Daniel Lucrédio · Tassio Vale · Eduardo Santana de Almeida · Silvio Romero de Lemos Meira

Published online: 1 October 2011

Abstract Duplicate bug report entries in bug trackers have a negative impact on software maintenance and evolution. This is due, among other factors, to the increased time spent on report analysis and validation, which in some cases takes over 20 min. Therefore, a considerable amount of time is lost in duplicate bug report analysis. In order to understand the possible factors that cause bug report duplication and its impact on software development, this paper presents an exploratory study in which bug tracking data from private and open source projects were analyzed. The results show, for example, that all projects we investigated had duplicate bug reports and a considerable amount of time was wasted by this duplication. Furthermore, features such as project lifetime, staff size, and the number of bug reports do not seem to be significant factors for duplication, while others, such as

Y. C. Cavalcanti (✉) · T. Vale · S. R. de Lemos Meira
Center for Informatics, Federal University of Pernambuco—CIn/UFPE, Pernambuco, Brazil
e-mail: ycc@cin.ufpe.br

T. Vale
e-mail: ceac@cin.ufpe.br

S. R. de Lemos Meira
e-mail: srlm@cin.ufpe.br

Y. C. Cavalcanti · P. A. da Mota Silveira Neto · D. Lucrédio · T. Vale · E. S. de Almeida · S. R. de Lemos Meira
Reuse in Software Engineering—RiSE, Recife, Brazil
e-mail: pamsn@cin.ufpe.br

D. Lucrédio
e-mail: daniel@dc.ufscar.br

E. S. de Almeida
e-mail: esa@dcc.ufba.br

D. Lucrédio
Computing Department, Federal University of São Carlos—DC/UFSCar, São Carlos, Brazil

E. S. de Almeida
Computer Science Department, Federal University of Bahia—DCC/UFBA, Bahia, Brazil

the submitters' profile and the number of submitters, do seem to influence the bug report duplication.

Keywords Bug reports · Bug tracker · Bug reports duplication · Exploratory study · Software configuration management

1 Introduction

Aiming to improve 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* (Johnson and Dubois 2003; Serrano and Ciordia 2005). A bug report[1] is defined as a software artifact that describes some defect, enhancement, change request, or an issue in general, which is submitted to a bug tracker by developers, testers, or even users. Such tracking systems are useful because software changes can be identified and quickly reported to developers Anvik et al. (2005). As they are stored in a database, they can serve as important project documentation, helping in tasks such as change impact analysis, effort estimation (Canfora and Cerulo 2005; Song et al. 2006; Weiss et al. 2007), software evolution and traceability (Fischer et al. 2003a, b; D'Ambros and Lanza 2006); Koponen and Lintula 2006).

Despite the benefits, there are some challenges in using bug trackers, such as the dynamic assignment of bug reports (Anvik et al. 2006; Canfora and Cerulo 2006; Anvik and Murphy 2007; Jeong et al. 2009), the quality of bug report descriptions (Ko et al. 2006; Bettenburg et al. 2007, 2008a, b; Castro et al. 2008), and the detection of duplicate bug reports (Podgurski et al. 2003; Anvik et al. 2005; Hiew 2006; Runeson et al. 2007; Jalbert and Weimer 2008; Wang et al. 2008; Bettenburg et al. 2008a, b). This work is focused on the last of these, the *bug report duplication problem*. This problem is characterized by the potential submission of two or more bug reports describing the same change request (issue). The main consequence of this problem is the extra effort necessary to manage these duplicates. For example, the staff responsible for the search and analysis of bug reports must spend a considerable amount of time checking whether a given bug report has not been submitted already. To show the extent of this problem, recent work (Anvik et al. 2005; Runeson et al. 2007) has shown that between 10 and 30% of a bug report repository is composed of duplicate bug reports.

The impact of duplicate bug reports can be seen in almost all maintenance and evolution activities. Generally, the maintenance and evolution process involves identifying, reporting (as bug reports), and verifying the impact of change requests; planning their implementation (i.e., to determine which release will implement the changes), implementing such changes and, finally, assembling a new release that includes the changes Sommerville (2007). Figure 1 gives an idea of the overall maintenance and evolution process.

In the first step, people responsible for identifying change requests must search and analyze previous bug reports in order to avoid the submission of duplicates. Failure to do so will burden all subsequent steps with unnecessary rework. The problem is that this is not uncommon, and many bug report repositories have duplicates in them.

[1] In some environments, the term *bug report* is replaced by *change request*, *request for enhancements*, *issue*, *ticket*, or just *bug*.

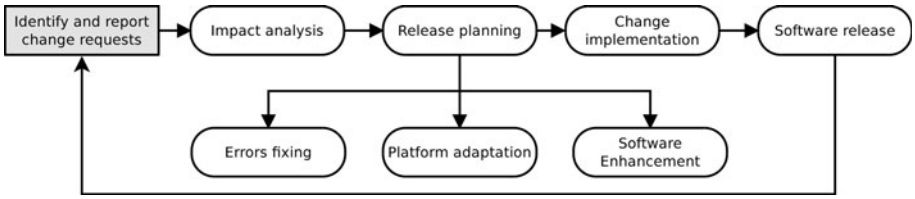

Fig. 1 Continuous software maintenance and evolution. Adapted from Sommerville (2007)

As a consequence of the inevitability of such duplicates 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. If the duplicates could be avoided at the first stage, all this time spent on search and analysis could be saved for other tasks.

To circumvent this problem, some organizations maintain a team with a special task of analyzing incoming bug reports, called CCB (Change Control Board). The CCB analyzes 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 process. However, even in the presence of a CCB, some duplicates remain unidentified Anvik et al. (2005). There are many reasons for that, such as lack of attention or experience of the CCB, poor analysis due to the volume of bug reports submitted per day; inadequate tools for the search and analysis of bug reports or even the lack of knowledge of the bug reports repository.

Furthermore, the cost of a duplicate bug report that passes through the entire software life cycle, without being identified at an early stage, is very hard to measure. Basically, the cost of the time lost with duplicates is transferred to everyone involved directly or indirectly with the software being developed, including testers, developers, managers, partners, and even customers.

Although recent work (Anvik et al. 2005; Runeson et al. 2007) has shown that bug report duplication is a critical problem, we found no answers in the literature to the following questions: What are the factors that cause the duplication problem and what are the main consequences of the problem? The answers to these questions would help us to understand the problem and to propose effective solutions. In this context, this paper presents an exploratory study (Wohlin et al. 2000) of bug repositories and the bug report search and analysis activities. The goal is to understand the possible factors that could cause bug report duplication and its impact on software development. This study is a continuation of the work presented in (Cavalcanti et al. 2010a, b). Both private and open source projects were used in the study, and a number of variables from the projects were analyzed.

The GQM approach (Basili et al. 1986) was taken to define and guide this exploratory study. The GQM consists of the object of the study, or goal, the questions to be answered, and the related metrics that must be used to answer the questions in a measurable way. For some metrics, we derived a baseline from related work and discussions, while others—for which no previous data were available—were explored in this work in order to establish baselines for future studies. Additionally, for some metrics, we provided some hypothesis testing and statistical correlation analysis.

The remainder of this paper is structured as follows: Sect. 2 presents the definition of the study, in terms of goals, questions, and metrics (GQM); Sect. 3 describes the projects and data used in the study. In Sect. 4, the data collection process is described, and in Sect. 5, the

analysis and interpretation of the results are presented. Section 6 presents the threats to the validity of the study, and Sects. 7, 8, and 9 present some advice to avoid duplicate bug reports, related work, and conclusion, respectively.

2 Study definition

2.1 Goal

As mentioned previously, this study used the GQM method (Basili et al. 1986). The goal of this study was to analyze bug repositories and the activities of searching and analyzing bug reports with the purpose of understanding them with respect to the possible factors that could have an impact on the duplication problem and their consequences for software development projects, from the researcher's point of view, in the context of software development projects. On the basis of related work, discussion with experts and feeling about what would be useful to characterize the duplication problem, the following questions were defined.

2.2 Questions and metrics

Question 1 *Do the analyzed projects have an appreciable number of duplicate bug reports?* This is the starting point of this study. Although the literature reports that there is a considerable amount of bug duplication in most projects, if the projects analyzed in this study do not have enough duplicate bug reports, the other questions cannot be answered properly. Thus, we investigated the projects to find out whether they have duplicate bug reports in their repositories and whether the number of duplicates is large enough to cause problems. In order to answer this question, the following metric was defined:

M_1: **percentage of duplicate bug reports in software development projects** This can be measured by analyzing the status of the bug reports in the repository. For example, if the status of a bug report is defined by the keyword DUPLICATE, then it counts as a duplicate. According to (Anvik et al. 2005; Runeson et al. 2007), we can expect that 20% of bug reports in bug repositories are duplicates, on average.

Question 2 *Is the submitters' productivity being affected by the bug report duplication problem?* As mentioned before, duplicate bug reports have side effects, such as extra time for analysis. In this context, the productivity is measured in terms of time needed to perform bug tracking activities, such as search and analysis of bug reports. Three metrics were used to understand this question:

M_2: **time spent in searching and analyzing bug reports before opening a new bug report** This time is measured in minutes and is counted from the moment a submitter begins to investigate a bug report until he/she decides whether it is new or duplicate. On the basis of informal interviews with submitters from C.E.S.A.R.[2], our private company partner, we expect values higher than 10 min for this metric. However, we must consider such values as potentially biased, since they were not empirically measured;
M_3: **ratio between the average time to resolve duplicate bug reports and average time to resolve valid bug reports** For example, if valid bug reports take x days on

[2] Recife center for advanced studies and systems. http://www.cesar.org.br.

average to be resolved, then we want to know the percentage of x that is necessary to solve duplicate bug reports. A duplicate bug report is considered solved when it is identified as duplicate and its state on the bug tracker state machine becomes CLOSED. Thus, to calculate M_3, we first divide it into two sub-metrics: (M_3') the average time (in days) to resolve duplicates; and (M_3'') the average time (in days) to resolve valid bug reports. Then, we calculate the ratio $M_3 = M_3'/M_3''$. To compute such averages (M_3' and M_3''), we investigated the lifetime of each bug report. We did not find any previous data to serve as a baseline for this metric;

M_4: **average frequency of bug reports per day**. This is the average number of bug reports that are submitted per day to the projects' repositories. This measure is calculated by dividing the number of bug reports by the total number of days that include them. Using this metric together with metric M_2, we can analyze how much time is being spent per day on search and analysis of bug reports. We did not find any previous data to serve as a baseline for this metric;

Question 3 *How are the relationships between master bug reports and duplicate bug reports characterized?* Here, we consider three types of bug report in a bug repository (Bettenburg et al. 2008a, b): (a) *unique bug report*, (b) *master bug report,* and (c) *duplicate bug report*. A report is classified as unique if it is the only one to describe an issue. If a set of reports describes the same issue, the first entry will be declared the master bug report and the others defined as duplicate bug reports. The process of binding masters to their respective duplicates is called *bug report grouping*. It is important to know how the groups are characterized, because this can guide us in choosing adequate techniques to solve the bug report duplication problem. For example, if there are only a few bug report groups, machine learning techniques may not be appropriate to tackle the duplication problem, because there are insufficient data to train the algorithm.

M_5: **distribution of grouping types** We counted two types of grouping: (M_5') master bug reports that have only one duplicate bug report, known as *one–one* relationships, and (M_5'') master bug reports that have more than one duplicate bug report, known as *one–many* relationships. We could define more levels of relationship, but we believe that these two are sufficient for our analysis.

Question 4 *Is there a common vocabulary for bug report descriptions?* The answer to this question is important because it is believed that a controlled vocabulary could help to prevent duplicate bug reports (Lancaster 1986). For example, with a well-defined vocabulary, the submitters could perform a better search using keywords closer to those present in the new bug report. A controlled vocabulary, in this case, could be a document template that guides the description of bug reports.

M_6: **percentage of common words shared in a bug report group that relates to the same issue** In order to gather relevant results, this metric does not count English language *stop words*[3]. We analyzed only summary and long description fields of the bug

[3] *Stop words* are words that do not improve the searches in information retrieval systems. Examples of stop words are *the*, *of*, *should*, *themselves*, etc. A comprehensive list of common stop words can be found in: http://www.ranks.nl/resources/stopwords.html.

reports. However, it is important to highlight that this metric holds a threat to validity: if there are a few large groups of bug reports related to the same issue, sharing many common words, the overall value would increase. As a result, a high percentage of shared words would appear, even if most groups in the repository did not share a common vocabulary. No previous data were found to serve as a baseline for this metric;

Question 5 *Does the type of a bug report influence the number of duplicates?* There are two main types of bug report: *enhancements* and *defects*. Enhancements are normally requests made by users and developers who want new or improved features on some product. Defects are errors or malfunctions reported by users and normally need to be corrected as soon as possible. Intuitively, it may be argued that defects have more duplicates, because it is more likely that two users will notice the same defect—and report it—than that the same enhancement potential is perceived and requested simultaneously by two different users. Such a difference would mean that defect bug reports require a more careful analysis than enhancement reports, for example.

M_7: **Duplication ratio = duplicate bug reports/total bug reports** For each bug report type—enhancement and defect—we calculated the ratio of the number of duplicates to total bug reports. If this ratio is larger for defects, then defects cause more duplicates than enhancements.

Question 6 *What factors can have an impact on the bug report duplication problem?* For the chosen projects, we adopted six variables that we believe could be factors for duplication or, at least, have some indirect relation to it. These variables are as follows:

Staff size This variable is related to the number of people involved in project development. We assumed the number of developers to be equal to the number of people assigned to resolve a bug report. Related work (Anvik et al. 2005) had shown that projects with a large staff had many duplicates, so we wanted to find out whether this was also true for projects with a small staff;

Number of submitters This the number of people who submitted bug reports in the period we collected the data. As in (Anvik et al. 2005), the analyzed projects had many submitters, but we would like to know whether duplicates are also present in projects with few submitters;

Software size This variable is related to the number of lines of code (LOC) that a software project had at the time of data collection. Comments and blank lines were discarded. The values for this variable were obtained through the site http://www.ohloh.net. We believe that the number of LOC can influence the number of errors and, consequently, more bug reports would be submitted, increasing the chances of duplication;

Software lifetime The is the amount of time a software project has been in development, measured in years. We computed the lifetime of a project from the first bug report submission. Anvik et al. (2005) analyzed projects of 7 and 9 years, but we had to analyze projects with a range of lifetimes, to dinf out if it is a factor for bug report duplication;

Bug repository size This variable is related to the number of bug reports a project has in its bug repository. With this variable, we can analyze whether or not large bug repositories are more susceptible than small ones to the submission of duplicates. In Anvik et al. (2005), the bug repositories had a large number of bug reports, but we wanted to know whether duplicates are also present in small bug repositories. By large repositories, we mean those in the range of tens of thousands of bug reports;

Submitters' profile It is important to know what type of submitter profile is more susceptible to submitting duplicate bug reports. The values for this variable are *sporadic* (S), *average* (A), and *frequent* (F). *Sporadic* is a reporter who submitted at most 10 bug reports in the analyzed period, *average* is a person who submitted between 10 and 30 bug reports in the period, and *frequent* is a person who submitted more than 30 bug reports in the period. These values are not exact and can vary depending on the overall number of submitters and users. Furthermore, we did not find any previous work that could support us in defining such values.

3 Projects and data selection

To conduct this study, we considered projects from open source organizations and from one private organization. Six companies were contacted, but only one had data we could use in our study. In the other companies, the bug reports were not described directly by users, but reported by people from a call center responsible for handling customer requests.

Eight open source projects were chosen. For the private project, we chose bug reports from a project being developed at C.E.S.A.R. Regarding the open source projects, we collected all bug reports until the end of June 2008. For the private project, we collected all the bug reports from November 2006 to March 2008, which includes the entire life cycle for this project. More details about each project are presented in Table 1. Information about each open source project can easily be found in the web.

We tried to cover a range of usage characteristics when choosing these open source projects. We selected some projects focused on end users, such as Firefox, Evolution, Thunderbird, and Epiphany, each one with a different number of users. We also selected projects that are used both by end users and developers, such as Bugzilla. Finally, projects that are used only by developers, such as Eclipse, GCC, and Tomcat were also included in the study.

Bugzilla Bugzilla (http://www.mozilla.org) is a software application for bug report tracking. These projects are hosted by Mozilla Foundation.
Eclipse Eclipse (http://www.eclipse.org) is an open development platform that supports C, PHP, Java, Python, and other languages.
Epiphany This is the official web browser for, and hosted by, the Gnome desktop (http://www.gnome.org).

Table 1 Characteristics of projects

Project	Domain	Code size (LOC)	Staff size	Bugs	Lifetime
Bugzilla	Bug tracker	55 K	340	12,829	14
Eclipse	IDE	6.5 M	352	130,095	7
Epiphany	Browser	100 K	19	10,683	6
Evolution	E-mail client	1 M	156	72,646	11
Firefox	Browser	80 K	514	60,233	9
GCC	Compiler	4.2 M	285	35,797	9
Thunderbird	E-mail client	310 K	192	19,204	8
Tomcat	Application server	200 K	57	8,293	8
Private project	Mobile application	2 M	21	7,955	2

Evolution Evolution 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 Firefox is a popular web browser that runs on a variety of platforms (Windows, Linux, Mac etc). Firefox is also hosted by Mozilla.
GCC GCC (http://www.gcc.gnu.org) is a collection of compilers, including front ends for C, C++, Objective C, Fortran, Ada, and Java.
Thunderbird. It is a cross-platform desktop e-mail client hosted and developed by Mozilla Foundation.
Tomcat. Apache Tomcat (http://www.apache.org) is an implementation of the Java Servlet and JavaServer Pages technologies, developed by the Apache Foundation.
Private project. This is a private project being developed at C.E.S.A.R., which involves the development of applications for mobile devices. It is also a test center for this type of application. We chose this project because it was the biggest project that we had full access to.

4 Study execution

This study was performed at C.E.S.A.R. over 2 months. At the beginning of the study, some meetings were held with project managers and stakeholders from the private project. During these meetings, we had discussions about the bug report duplication problem, to understand how it was affecting the projects and how to mitigate it. These meetings gave us insights about how to approach the problem. In addition, we defined a set of instrumentation assets to be used in the study, such as scripts, time sheets, and questionnaires. These instruments are described as follows:

Scripts Python scripts were built to read the bug reports in XML format from each project. From these scripts, which are detailed later, we were able to obtain the values for metrics M_1, M_3–M_7. From these scripts, we also identified the 20 most active submitters from the analyzed projects. To do this, we counted the bug reports submitted by each one of them. We also used R-project[4] scripts to provide the hypothesis testing and correlation analysis. Metric M_2 was collected with time sheets and a questionnaire, which are explained next.

Time sheets Time sheets were used to collect the time employed by the staff of the private project to open bug reports (metric M_2). A total of four people filled out the time sheets during a period of 2 weeks. The time sheets were applied only to this project because we had access to it and could monitor this activity. For other projects, this information was gathered through a questionnaire, as described next.

Questionnaire The questionnaire was sent to the 20 most active submitters from each project. We tried to make the questionnaire as simple as possible. It asked about the time spent to search and analyze bug reports and whether any techniques were used to avoid duplicates.

In total, the questionnaire was sent by e-mail to 180 submitters, with a 2-weeks deadline for answer. Of these e-mails, 24 could not be delivered to the recipients and 141 did not respond to the questionnaire in time. Only 17 developers, from 7 projects, replied to the questionnaire: 1 person from Bugzilla; 1 from Evolution, 4 from Epiphany, 3 from Firefox,

[4] http://www.r-project.org/.

2 from GCC, 2 from Thunderbird, and 4 from the private project. This low number of answers is further discussed in the *Threats to Validity* section.

The submitters had between 2 and 8 years of experience in the project in which they participated. Moreover, 14 submitters said they spent from 5–10 min searching for bug reports, and only 3 submitters chose the answer 10–15 min. For the private project, this time was collected from the time sheets and fell between 20 and 30 min. As mentioned before, such numbers for open source projects can be very biased when the values are obtained through a questionnaire. Answers would be more precise if we could use time sheets as we did for the private project.

Thus, people in the private project spend more time searching than those in the other projects. This is probably because it is a project dedicated to testing, and testers are assessed and advised to try their best to avoid duplicate submissions. Furthermore, the testers are hired to perform such work.

Regarding the techniques used to avoid duplicates, although the projects do not have automatic techniques for detecting duplicates, some warnings and guidelines are shown in bug trackers to inform the submitter about the duplication problem. Some bug trackers also show a list of most submitted bug reports in order to minimize their resubmission.

4.1 Hypothesis testing and correlations

For some metrics, we performed only descriptive statistics to analyze the data, while for other metrics, where the amount of collected data was considerable, we tested some hypotheses and correlations between variables that would be useful to understand the problem. For the latter analysis, we used *bootstrap* techniques Efron and Tibshirani (1993).

Bootstrap techniques are used when the problem being addressed cannot be solved by classical methods or when the conditions for applying such methods are not valid. The idea behind the bootstrap is to use new random samples of the initial observations to provide more fine-grained statistical inference. In other words, we selected new samples by randomizing the initial set of observations. Under certain regularity conditions, the theory shows that the distribution of the parameter sequence obtained converges to the actual distribution of the parameters.

After obtaining the bootstrap samples, we carried out the Kolmogorov–Smirnov adherence test that checks the normality of the sample distribution. After performing the Kolmogorov–Smirnov test, we applied Student's t test. We used 5,000 samples by resampling the initial observation sample using the bootstrap techniques.

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 data from the private project for reasons of confidentiality.

4.2 Data gathering

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

```
<bugzilla>
  <bug>
      <bug_id>391316</bug_id>
      <creation_ts>2007-08-07 20:23 PST</creation_ts>
      <short_desc>Add FTP support to /short_desc>
      <delta_ts>2007-08-07 20:23:16 PST</delta_ts>

      <!–other fields could come here –>

      <reporter>rflint@mozilla.com</reporter>
      <assigned_to>rflint@mozilla.com</assigned_to>
      <qa_contact>microsummaries@firefox.bugs</qa_contact>
      <long_desc isprivate="0">
        <who>rflint@mozilla.com</who>
        <bug_when>2007-08-07 20:23:16 PST</bug_when>
        <thetext> </thetext>
      </long_desc>
  </bug>
</bugzilla>
```

5 Analysis and interpretation

This section analyzes the questions defined in Sect. 2, according to the metrics obtained from each project. It also presents a descriptive analysis of the data.

Question 1 Do the analyzed projects have an appreciable number of duplicate bug reports?

The values for M_1 in Fig. 2 show a high percentage of duplicates for eight projects. The exception was the Tomcat project. Only three projects were below than expected (Eclipse, Tomcat, and GCC). However, the overall average of duplicates exceeded our expectations in 8.1%.

Hypothesis testing The null hypothesis is that the duplication rate is less or equal than 20%, while the alternative hypothesis is that it is more than 20%. The average duplication of the original sample is 28.71% and the bootstrap sample is 28.73% (median 28.62%). Applying a 99% confidence interval, we have a lower limit of 18.28% and an upper limit of 38.95%, as illustrated in the histogram of Fig. 3. Moreover, the Kolmogorov–Smirnov test

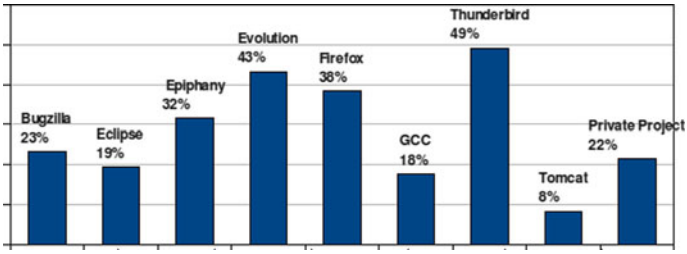

Fig. 2 M_1: percentage of duplicate bug reports for each project

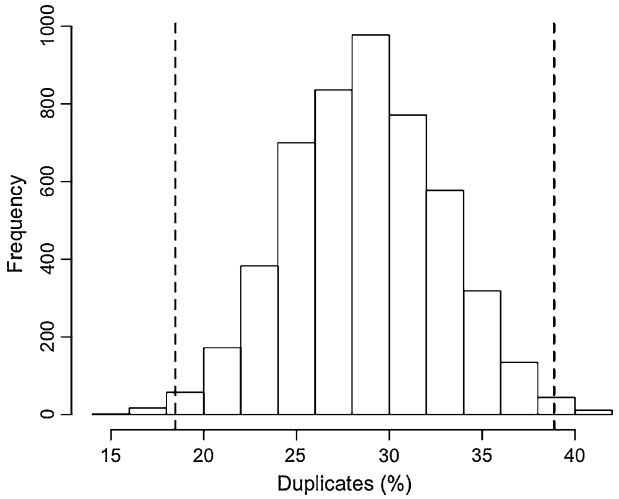

Fig. 3 Bootstrap histogram for duplication

Table 2 Hypothesis testing with bootstrap for duplication

Original sample average	28.71	
Bootstrap sample average	28.72	SE = 4.11
Confidence interval	99%	
Lower limit	18.12	
Upper limit	38.41	
Kolmogorov–Smirnov	*P* value <2.2e−16	0.9283 (a normal distribution)
Stundent's *t*	*P*value <2.2e−16	Null hypothesis rejected

indicated a *P* value of 0.9283, which means that the sample follows the normal distribution.

The Student's *t* test indicated a *P* value of `2.2e-16`, which rejects the null hypothesis. Thus, we considered that the number of duplicates was appreciable, and therefore, we could continue our study. The results are summarized in Table 2.

Question 2 Is the submitters' productivity being affected by the bug report duplication problem?

To answer this question, we analyzed metrics M_2, M_3 (Fig. 4), and M_4 (Table 5). Every person who answered our questionnaire stated that they were concerned with duplicates and that they performed search and analysis before submitting a bug report. For metric M_2, people from five projects stated that they spent 5–15 min with search and analysis, while people from one project stated that they spent 5–10 and 20–30 min in another. We had no answers for the other two projects.

Figure 5 According to metric M_2, the average search time was 12.5 min (this value was computed using the individual responses from reporters). Metric M_4 shows an average of 231.5 bug reports submitted per day.

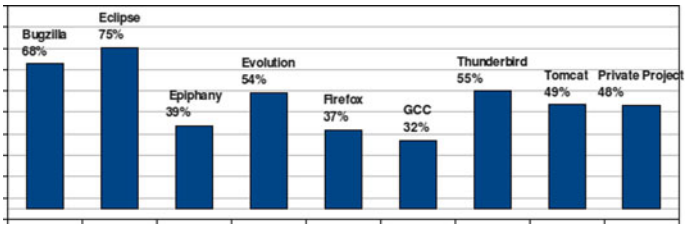

Fig. 4 M_3: ratio of the average time to resolve duplicate bug reports to the average time to resolve valid bug reports

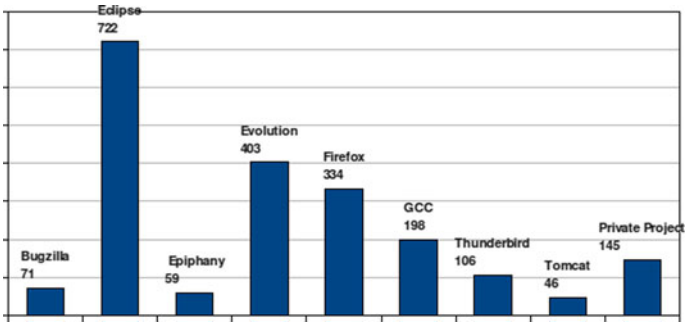

Fig. 5 M_4: average frequency of bug reports per day

If we assume that the people who answered the questionnaire are those responsible for most bug report submissions, we have an approximate average of 48 man-h[5] that must be spent per day in the search for duplicates. If we could reduce this time in half, for example using some techniques to suggest similar bug reports, we would save $\approx$24 man-h per day, which is considerable.

Another important point can be drawn after an analysis of M_3: duplicates are resolved using half the time needed to resolve a valid bug report, on average. Thus, the reduction in the number of duplicate bug reports that are inserted into a repository would save valuable time, which could be used to resolve valid bug reports. For example, for every two duplicates avoided, a valid bug report could be resolved. However, it is important to note that we are considering the time to resolve a bug report to be the elapsed time it takes to be closed, that is, when its classification changes to CLOSED in the bug tracker. An analysis of the time spent with each individual bug report is required to determine the actual effort required for analysis.

Hypothesis testing Our null hypothesis for this question is that the average time for bug report analysis is less than or equal 10 min, while the alternative hypothesis is that it is more than 10 min. Thus, we performed the same calculation presented in Question 1 for hypothesis testing and summarized it in Table 3. As it can be observed, the null hypothesis was rejected.

Only these values for descriptive statistics and hypothesis testing are enough to demonstrate how the bug report duplication problem can affect productivity, which is the goal of Question 2. However, these are too simplistic, because bug report search and analysis

[5] This number was calculated by multiplying the average of bug reports per day by the average time spent with search and analysis of bug reports: (231.5 *bugs* $\times$ 12.5 *min*)/60–48 *man-h*.

Table 3 Hypothesis testing with bootstrap for analysis time

Original sample average	13.88	
Bootstrap sample average	13.89	SE = 2.29
Confidence interval	99%	
Lower limit	8.88	
Upper limit	20	
Kolmogorov–Smirnov	P value <2.2e−16	0.0657 (a normal distribution)
Stundent's t	P value <2.2e−16	Null hypothesis rejected

are just the tip of the iceberg. There are many tasks spread over the entire software life cycle that are influenced by the problem. Further analysis is needed to measure the full impact of duplicates.

Question 3 How are the relationships between master bug reports and duplicate bug reports characterized?

Before we analyze this question, it is necessary to detail some particularities of bug report groups. As described by Bettenburg et al. (2008), when a duplicate is identified, it is marked with the DUPLICATE resolution tag and the ID of the master bug report is recorded. This process of identifying duplicates generates groups of bug reports, where each group is concerned with a specific problem. In other words, each group has a master bug report and a set of duplicates related to it. We refer to such groups as *duplicate groups*.

For the purposes of this study, there are two types of bug report group: (M_5') *one–one*, a master bug report with a single duplicate; or (M_5'') *one–many*, a master bug report with multiple duplicates.

As the bug report database grows, it is natural for people to start to forget which master bug report ID should be selected for a new duplicate report, generating what we call *false segregation of duplicate groups*. The false segregation of duplicate groups is characterized by the existence of different duplicate groups related to the same issue.

The false segregation of duplicate groups leads to information loss, since the complete set of information is spread over different groups. It also makes the identification of new duplicates more difficult. Thus, in order to analyze Question 3, we should first unify these false segregated duplicate groups.

The unification of such groups is also the subject of another study Bettenburg et al. (2008a, b). However, in Bettenburg et al. (2008a, b), the master bug report was defined by the developers. In our approach, we decided to elect the earliest bug report as master, because we verified that these are the bug reports with more comments and are probably the most important of the group.

In order to unify the false segregated duplicate groups, we followed two steps: (a) we first identified all duplicate groups, and next (b), we unified all common groups. To perform the first step, we used fields bug_id and dup_id present in the XML files to create dictionary structures where the key is the master bug report and the values are the duplicates. At the end of this step, we obtained a collection of dictionary structures like the one shown in Table 4.

As it can be seen in dictionary structures from Table 4, bug report Bug 34423 is present in two groups, 1 and 2. Thus, these groups are related to the same issue and should be merged to become a single group. The result of the merge is shown in Table 5. Note that

Table 4 Initial dictionary structures

No.	Key (master bug reports)	Values (duplicate bugreports)
1	Bug 34423	Bug 43454, 87849, 756394, 843974
2	Bug 49812	Bug 34423, 54329
3	Bug 98986	Bug 36748, 4785937, 563849, 121234
4	Bug 88986	Bug 237484

Table 5 Dictionary structures after merge

#	Key (master bug reports)	Vaues (duplicate bugreports)
1	Bug 34423	Bug 43454, 87849, 756394, 843974, 49812, 54329
2	Bug 98986	Bug 36748, 4785937, 563849, 121234
3	Bug 88986	Bug 237484

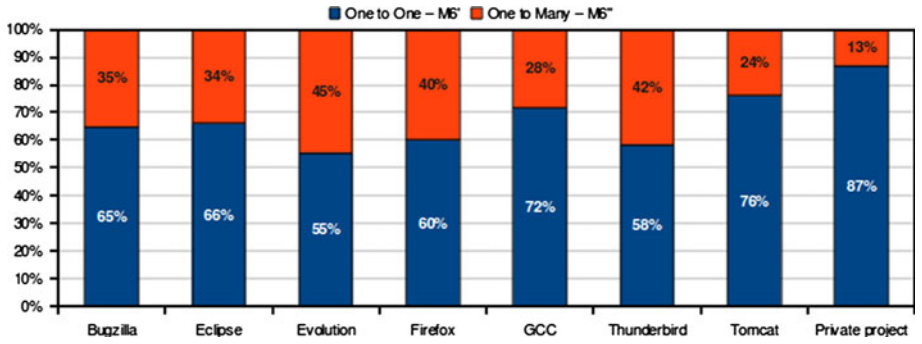

Fig. 6 Characterization of bug report groups

the other two bug reports (Bug 49812 and 54329) from old group 2 were moved to group 1. The same happens in many duplicate groups. The second step is responsible for unifying the duplicate groups related to the same issue, thus performing the unification of the false segregated duplicate groups. When all duplicate groups are unified, the earliest bug report of each group is defined as its master and the others become its duplicates.

Repositories with most groups classified as one–many may take advantage of techniques such as clustering or machine learning, as presented in Hiew (2006). These techniques work well when there is enough data to allow proper learning. For example, if we apply a machine learning algorithm to recommend possible duplicates when a reporter tries to open a new bug report, first this algorithm must look into all existing bug reports in order to extract some patterns. Only then it can recommend bug reports that match the pattern of the new one. Thus, if there are many bug report groups, there are better chances that the algorithm extracts relevant patterns and then correctly classifies incoming bug reports as duplicates.

As Fig. 6 shows, all studied projects have more than 60% of bug report groups characterized as one–one. As mentioned before, machine learning and clustering techniques are not adequate in this case. This suggests the need for techniques that work with repositories where one–one relationships predominate.

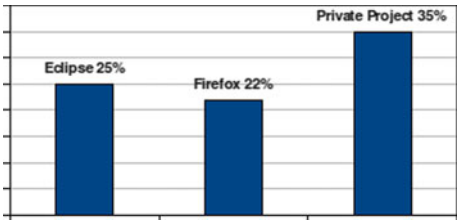

Fig. 7 M_6: percentage of common words shared in a bug report group

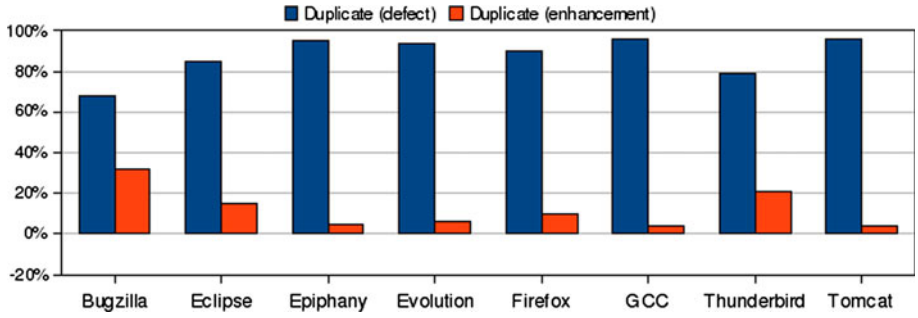

Fig. 8 Absolute duplication ratio

Hypothesis testing We also did not provide hypothesis testing for this metric, because we did not have a baseline from which we could determine the null hypothesis.

Question 4 Is there a common vocabulary for bug report descriptions?

We analyzed the descriptions of the bug reports from Eclipse, Firefox, and the private project. We were unable to examine the other projects' descriptions because their duplicate bug reports did not specify their masters, and therefore, the groups could not be identified. As can be seen in Fig. 7, the open source projects did not have a considerable difference for metric M_6, while the bug reports from the private project share more common words than the others. However, we do not have a baseline value for this metric to determine whether these values are low or high.

We believe that the value for the private project was higher because its environment was more restricted and controlled due to the reduced staff size and number of submitters. In contrast, in the more uncontrolled environments of the distributed open source projects, any user can submit a bug report, and thus it seems natural that a wider variety of words was used to describe a single issue. Also, the bug reports in the private project are described using a template. This suggests that in open source projects, it may be a good idea to restrict the vocabulary in the bug report repository somehow.

It is important to observe that some threats involving our algorithms can influence these results. For example, we did not filter stack traces or source code present in the bug trackers. These kinds of information can increase the number of common words shared among bug reports. Another issue is that some of the bug reports can be present in the wrong groups if their master bug report was incorrectly specified during the analysis.

Hypothesis testing We did not provide hypothesis testing for this metric because we did not have an expected value from which we could determine the null hypothesis.

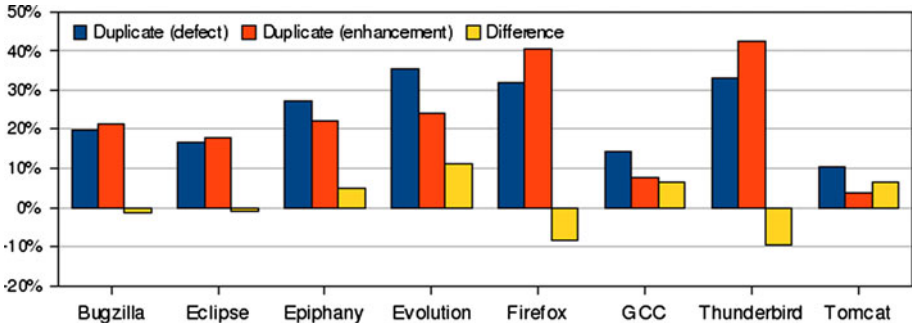

Fig. 9 Duplication ratio

Question 5 Does the type of a bug report influence the number of duplicates?

For each bug report type, we may conclude that there are much more defect duplicates than enhancement duplicates, as shows Fig. 8.

However, an analysis of the distribution of the bug report types showed that more than 87% of the bug reports are related to defects. Thus, it is natural that there are more defect duplicates in the repository. We needed to know whether this difference also shows up when we calculate the relative percentage of duplicate bug reports for each bug report type. Figure 9 shows the relative duplication ratio for defects and enhancements, and their difference in terms of percentages, for each open source project. We did not compute it for the private project because all its bug reports are about defects.

As we can see, the ratios are distributed without major differences among the projects. Some projects have relatively more duplicate defect reports than duplicate enhancement requests, while in other projects, the opposite is observed. The mean of the sum of all differences is 3.23%, which indicates that duplicate bug reports for defects are slightly more frequent than duplicate bug reports for enhancements. But such a small value is not enough to conclude that duplication is influenced by the bug report type. It seems more likely that the duplication problem does not depend on the bug report type.

Question 6 What factors can have an impact on the bug report duplication problem?

Number of submitters and staff size Figure 10 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 lifetime of the project, because we considered, in that case, the year the first duplicate bug report was submitted.

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

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

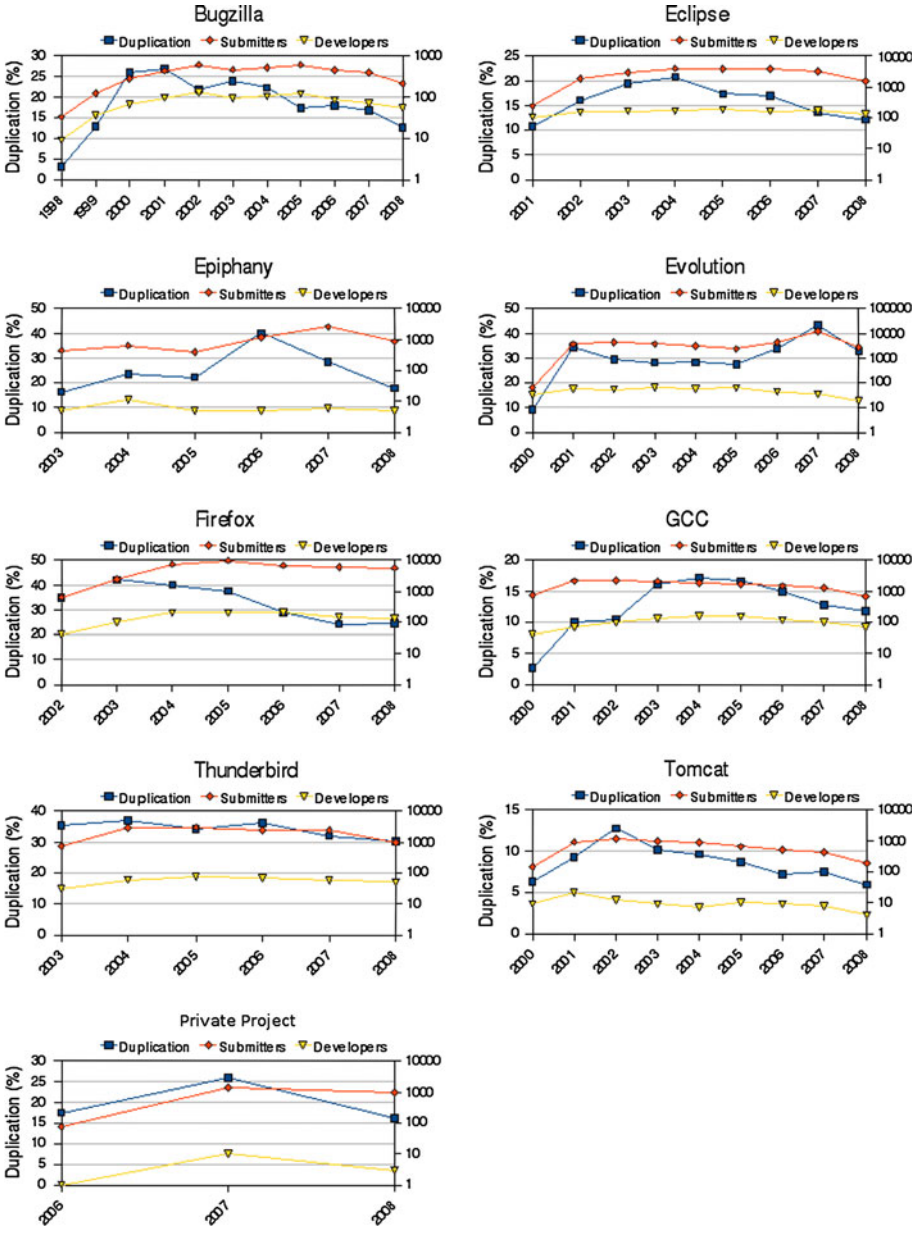

Fig. 10 Duplication versus staff size versus submitters. Data reduced using log 10 due to the large range of values

has 14 years of lifetime, but the first duplicate bug report appeared only in 1998 when the first release was made available to users (version 2.0). The same also seems to happen for major changes and new releases, like years 2006 and 2007 for the Evolution project.

The staff size did not change considerably in almost all projects. It also does not seem to have a direct relationship with the number of duplicates. In contrast, the number of

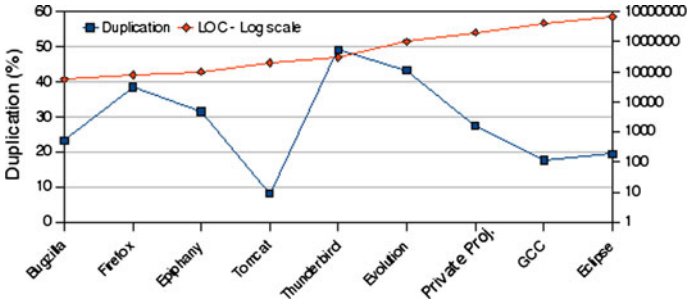

Fig. 11 Duplication and LOC. The *x*-axis is ordered from the smaller number of LOC to the biggest

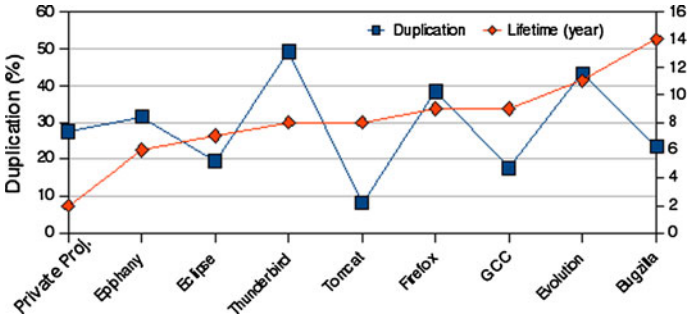

Fig. 12 Duplication and lifetime

developers seems to be connected with the number of submitters. We also observed that an average of 76% of the developers (per project) are also bug report submitters. From these 76%, an average of 13 submitters are among the 20 most active. So, when there is a drop in the number of developers, there is also a drop in the number of submitters.

Software size According to Fig. 11, there is no noticeable pattern that leads us to conclude that there is some relationship between the number of LOC and the number of duplicates. For example, the Eclipse project has the greatest number of LOC, but 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%.

Software lifetime As shown in Fig. 12, we observed that the lifetime is not a factor for the duplication problem. Projects with longer lifetimes do not necessarily have more duplicate bug reports than projects with shorter lifetimes. The Thunderbird and Tomcat projects are good examples: both have similar lifetimes, but differ sharply on the number of duplicates. Another example is the Bugzilla and the private project: the former is 12 years older than the latter, but it has about 10% less duplicates.

Bug repository size The number of bug reports in the repository is not a causing factor fo the duplication problem, as shown in Fig. 13. This observation contradicts our initial thoughts. We believed that the larger the bug report repository, the more difficult it is to find similar bug reports. However, we must take into account that as the number of bug reports increases, the knowledge of the submitters on the repository also increases, which balances the repository growth factor.

Submitter profile Figure 14 shows the contribution of each type of submitter to the number of duplicates in each project. As it can be seen, most of the duplicate bug reports

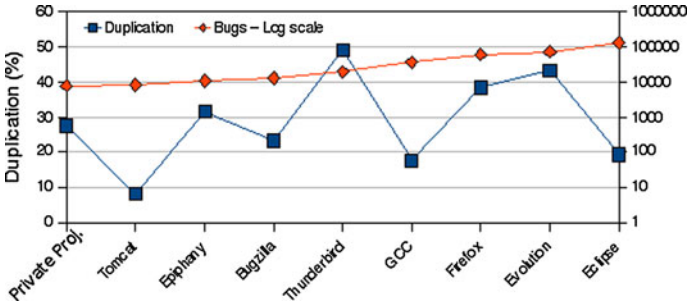

Fig. 13 Duplication and repository size

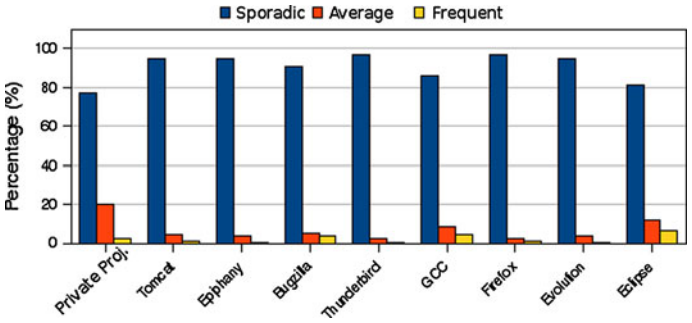

Fig. 14 The duplication percentage according to each submitter profile

are submitted by sporadic submitters, followed by average and frequent submitters. The average and frequent submitters contribute very little to the duplication problem, because they are more experienced in the project, with good knowledge of the repository.

We noticed that sporadic submitters are more prone to submit duplicate bug reports. One possible explanation could be the fact that they probably have insufficient knowledge of 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 and are not aware that a thorough search must be carried out prior to submission. Moreover, according to our analysis, most sporadic submitters are responsible for at most 1 or 2 bug reports during the entire lifetime of the project.

Correlation analysis In this section, we describe the correlation analysis carried out in order to determine whether the characteristics of the software projects discussed so far are statistically related to the bug duplication problem. We used *Pearson* correlation to perform the calculations. Table 6 summarizes the results.

Except for software size, the correlation analysis is in agreement with the discussion we presented for each characteristic investigated in Question 6. In particular, the submitter profile, specially sporadic submitters, which was identified as the most influent factor to the duplication problem, had a strong correlation according to our analysis.

5.1 Main findings of the study

This study was conducted with the goal of understanding which characteristics of software projects can be factors for bug report duplication. We also wanted to know whether the

Table 6 Analysis of correlations for the characteristics of the studied projects

Correlation	*P* value	Intensity
Number of submitters versus duplication	0.51	Moderate
Staff size versus duplication	0.09	Weak
Software size versus duplication	0.35	Weak/moderate
Software lifetime versus duplication	0.06	Weak
Bug repository size versus duplication	0.04	Weak
Sporadic submitters versus duplication	0.84	Strong
Average submitters versus duplication	0.60	Moderate
Frequent submitters versus duplication	0.43	Weak

problem actually exists in the examined projects and where its impact on development would be perceived. The main findings of this study were as follows:

(a) All analyzed projects have duplicate bug reports in their repositories, and thus 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 submitter productivity in the analyzed projects is being affected by the bug reports duplication problem. According to the estimate of this study (taking into account all projects), almost 48 man-h 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 makes it more difficult to identify duplicates. Moreover, none of the projects have explicitly defined a vocabulary to control bug report descriptions. On the other hand, the private project was the only one to use a template to describe the reports, having also more shared words. This indicates that a template could be used to increase the number of shared words, consequently facilitating the identification of duplicates;
(d) We observed that submitters from the private project are more likely to submit reports using a common vocabulary than submitters from open source projects;
(e) For the three projects where bug report grouping was analyzed, more than 80% of the groups were composed by one–one relationships. If this same behavior is observed in other projects, we may conclude that clustering and machine learning 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 (defect or enhancement);
(g) Software size seems to be a weak/moderate factor for the duplication problem, although we could not identify such influence before we performed a correlation analysis;
(h) The size of the repository does not seem to be a factor for duplication;
(i) Project lifetime does not seem to be a factor for duplication;
(j) Staff size does not seem to be a factor for duplication;
(k) The number of submitters seems to have a moderate influence on the duplication problem; and
(l) The profile of the submitters is a determining factor for the submission of duplicates. In particular, sporadic submitters seem to be more likely to submit 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 validity:

Gathering information from reporters We had difficulties to gather information from open source developers. We selected 180 developers to send the questionnaire, but we received only 17 responses. Such a small number of responses is a threat to the validity of this study.

There could be many reasons for this low number of replies, such as invalid e-mails, lack of interest, or even the fact that people create an e-mail just to have access to the repository system and never actually read it. Thus, alternative methods for gathering such information must be developed.

Furthermore, the way the time spent on search and analysis of bug reports was collected can be very biased, because it was self-reported.

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 Since this is an initial characterization study about the duplication problem, we used descriptive statistics for all metrics and hypothesis testing and correlation analysis for some of them. However, we believe that this study is now a starting point for others. New approaches should formalize and test new hypotheses based on our initial results.

For a future study, it could be interesting to collect data from more projects and group them into categories such as projects with highly skilled people, projects with small, medium, and high number of end users (and developers), and so on. An analysis and interpretation could be based on such groups to draw more interesting conclusions.

Negative results We investigated some aspects of the projects that could be possible causes for the duplication problem, but most of them were not confirmed. However, these should not be discarded immediately. More in-depth analysis of the negative results must be provided to increase the validity of the conclusion.

Projects population We analyzed eight open source projects and one 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, from our experience and discussions with practitioners from our partner company, we believe that private projects that have similar characteristics (i.e., LOC, submitters, test center) are also being similarly affected by the bug report duplication problem.

Correlation versus causation We must not confound correlation with causation. The possible correlations identified 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 Advice to avoid duplicate bug reports

The findings of this study can help to establish some actions in order to improve the current practice on identification of duplicates. So, we listed below some advice that could be helpful:

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

One way to achieve the prediction of a profile is to look at how many bug reports a given person has submitted before. Another way is to check when the submitter last signed up into the bug repository. The bug repository administrator should decide which is better for his/her project.

Control the vocabulary and increase the number of fields in the bug report form If similar bug reports are described with different words, the identification of duplicates is more 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* (Ko et al. 2006; Wang et al. 2008) would facilitate the use of techniques to find duplicate entries based on such type of information.

Improve the search algorithms One of the best ways to reduce the number of duplicates is to improve the search algorithms. We believe that this is a key point for future research in this field. In this context, in an initial study, we have identified that bug reports with a high number of comments are also the ones for which more duplicates will be submitted. This is intuitive, because the most commented bug reports are those describing issues people are facing more frequently.

We have implemented a search engine that uses the number of comments to influence the search results. Initial tests indicate that such approach really seems to improve the identification of duplicates. This solution is simple to implement in the existing tools, since most of them provide a plug-in-based architecture.

The launchpad[6] tool, which has an integrated bug tracker, has recently implemented this feature. However, further empirical studies must be conducted to yield more precise results about this technique.

We understand that by reducing the number of duplicates, the time to search and analyze bug reports—which is the key point of the problem—will not necessarily be reduced too. However, it could help to avoid rework in other steps of the software maintenance and evolution process.

Improving the bug report analysis method The most common way to perform analysis of bug reports in order to avoid duplicates is to look at the reports one by one, browsing the entire repository. This is clearly a hard and time consuming work and not so efficient as we can see in the results of our observations. Thus, one way to improve the detection of duplicates is by improving the way in which bug reports are analyzed.

The Mozilla[7] projects have tried to improve the detection of duplicates by showing the *all time top 100* bug reports, as an alternative to manual analysis. In this way, before submitting a new bug report, the submitter should look at this list—which is the list with most frequently submitted issues—in order to reduce the probability of submitting a duplicate without having to browse the entire repository.

Another way to improve bug report analysis is by using computational techniques. For example, we believe that *information visualization* techniques (Card et al. 1999) could

[6] https://www.launchpad.net.

[7] http://www.mozilla.org/.

have a positive influence on this activity. One way to use such techniques would be to mix search and merging algorithms with graphical visualization, in such a way that the submitter could visualize multiple similar bug reports at the same time and decide whether a bug report is a duplicate or not. Our initial results with these techniques, presented in da Cunha et al. (2010), indicate that this is viable.

Education Finally, since sporadic submitters are the ones causing most duplication, we could be facing an education problem. Maybe these submitters, not being as experienced as the most frequent ones, are not completely aware of how problematic a duplicate bug report is, nor its consequences. A submission tutorial, video, or document would be a simple and relatively inexpensive solution, compared to the other suggestions above, to reduce this factor.

8 Related work

Podgurski et al. (2003) were the first to investigate the bug report duplication problem. However, the bug reports explored in their work were software failures automatically submitted when the software did not work properly. Such reports were composed of information (profile) about the state of the software at the time the failure occurred and possibly with the execution/call stack trace. Thus, Podgurski et al. proposed an automated support for classifying these reports in order to prioritize and diagnose their causes. The proposed approach used supervised and unsupervised pattern classification and multivariate visualization to group together bug reports with closely related causes.

After that, mainly because of the increasing adoption of bug report tracking tools, other researchers started to investigate bug reports described by humans using natural language. For example, Anvik et al. (2005) 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 on the duplication problem, we believe that it is complementary to Anvik's work, since we expanded the number of projects and variables analyzed in order to understand the problem.

Later, Hiew (2006) also investigated the duplication problem caused by the submission of natural language bug reports, which is closer to the scenario we are considering in this paper. Hiew proposed to group similar bug reports into *centroids*[8], so that it would be possible to compare incoming bug reports to the *centroid* with high similarity.

Runeson et al. (2007) addressed the problem of detecting duplicate bug reports using Natural Language Processing (NLP) techniques Feldman and Sanger (2007). One contribution of their work was the identification of two types of bug reports: (1) those that describe the same problem and (2) those that describe two different problems with the same cause. The former describes the same failure, generally using similar vocabulary, and the latter describes different failures and may use a different vocabulary. However, Runeson et al. (2007) restricted their approach to address type 1 only.

Wang et al. (2008) proposed an approach to mitigate the bug report duplication problem using NLP and execution information. The execution information is related to data about the software execution when the error occurred, such as method calls or state of variables. This type of data was combined with natural language data to improve recall.

Another research that attempts to detect duplicate bug reports was developed by Jalbert and Weimer (2008). They proposed a system to automatically classify duplicate bug

[8] A centroid, in this case, is a set of bug reports with high similarity among them.

reports as they arrive into the repository. Differently from other works, the system used surface features and textual semantics in conjunction with clustering techniques.

Bettenburg et al. (2008a, b) also addressed the problem of detecting duplicates, but they 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).

Finally, we cite the work of Sandusky et al. (2004). Although it was not originally proposed to avoid duplicates, we believe that their techniques could be applied with this purpose. The authors proposed a method to identify and visualize bug report networks. With such networks, it was possible to analyze relationships and dependencies among bug reports, which could facilitate the detection of duplicates.

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 projects. We believe that empirical studies are an important first step to understand the problem with a certain degree of confidence and define new directions for solutions. In this sense, these papers provide new insights into researchers, so that new proposals to solve the problem can be devised. We also try to help practitioners by compiling some advice to reduce the duplication problem based on our findings.

9 Conclusion

In this paper, we discussed the bug report duplication problem, in an exploratory study about the factors that may have impact on it. The study was performed using nine bug report repositories from open source and private projects, all of them with different characteristics, such as software size, staff, lifetime, number of bug reports, among others. Hypothesis testing, correlation analysis, and descriptive statistics were combined to derive our conclusions.

According to our analysis, in all nine projects, the bug report duplication problem was present. Furthermore, we found evidence indicating that the productivity is being affected by this problem. We also tried to identify which project characteristics may be influencing factors to the duplication problem and which do not seem to be related to it. The most influencing factor we observed was the submitter profile. In particular, sporadic submitters tend to be the ones submitting most duplicates. On the other hand, software size and project lifetime do not seem to have a direct influence on the problem, and that contradicts our initial intuition. The same is true for repository size.

We complement the findings of our study with some advice that could be useful to reduce the problem of duplicate bug reports. Indeed, we understand that it is not possible to reduce to zero the number of duplicates in a project or completely eliminate the rework caused by it. However, we believe that the effects of the duplication problem can be softened in some ways.

In future replications of this study, more private projects should be included and also more open source projects. It would be even better if the projects could be distributed into groups of related projects, sharing similar characteristics. Furthermore, new approaches should formalize and test hypotheses based on our initial results using, for example, probabilistic models and formal correlation techniques.

Acknowledgments This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES http://www.ines.org.br), funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08 and CNPq grants 305968/2010-6, 559997/2010-8, 474766/2010-1.

References

Anvik, J., & Murphy, G. C. (2007). Determining implementation expertise from bug reports. In *Proceedings of the fourth international workshop on mining soft. Repositories* (*MSR'07*). New York, NY: IEEE Press.

Anvik, J., Hiew, L., & Murphy, G. C. (2005). Coping with an open bug repository. In *Proceedings of the 2005 OOPSLA workshop on eclipse technology eXchange* (pp. 35–39). New York, NY: ACM Press. doi:10.1145/1117696.1117704.

Anvik, J., Hiew, L., & Murphy, G. C. (2006). Who should fix this bug? In *Proceedings of the 28th international conference on software engineering* (*ICSE'06*) (pp. 361–370). New York, NY : ACM Press.

Basili, V., Selby, R., & Hutchens, D. (1986). Experimentation in software engineering. *IEEE Transacations on Software Engineering, 12*(7), 733–743.

Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., & Zimmermann, T. (2007). Quality of bug reports in eclipse. In *Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange* (*eclipse '07*) (pp. 21–25). New York: ACM Press. doi:10.1145/1328279.1328284.

Bettenburg, N., Premraj, R., Zimmermann, T., & Kim, S. (2008a). Duplicate bug reports considered harmful? In *Proceedings of the international conference on software maintenance* (*ICSM'08*) (pp. 337–345). New York: IEEE Press.

Bettenburg, N., Premraj, R., Zimmermann, T., & Kim, S. (2008b). Extracting structural information from bug reports. In *Proceedings of the 2008 international workshop on mining software repositories* (*MSR'08*) (pp. 27–30). New York: ACM Press. doi:10.1145/1370750.1370757.

Canfora, G., & Cerulo, L. (2005). Impact analysis by mining software and change request repositories. In *Proceedings of the 11th IEEE international software metrics symposium* (*METRICS'05*) (p. 29). Washington, DC: IEEE Press. doi:10.1109/METRICS.2005.28.

Canfora, G., & Cerulo, L. (2006). Supporting change request assignment in open source development. In *Proceedings of the 2006 ACM symposium on applied computing* (*SAC'06*) (pp. 1767–1772). New York: ACM Press. doi:10.1145/1141277.1141693.

Card, S. K., Mackinlay, J. D., & Shneiderman, B. (1999). Readings in information visualization: Using vision to think. In C. Stuart, P. A. R. C. Xerox, & G. Jonathan (Eds.), *The Morgan Kaufmann series in interactive technologies*. MA, USA: Morgan Kaufmann.

Castro, M., Costa, M., & Martin, J. P. (2008). Better bug reporting with better privacy. In *Proceedings of the 13th international conference on architectural support for programming languages and operating systems* (*ASPLOS XIII*) (pp. 319–328). New York, NY: ACM Press. doi:10.1145/1346281.1346322.

Cavalcanti, Y. C., de Almeida, E. S., da Cunha, C. E. A., Lucrédio, D., & de Lemos Meira, S. R. (2010a). An initial study on the bug report duplication problem. In *Proceedings of the 14th European conference on software maintenance and reengineering* (CSMR'2010) (pp. 273–276). Madrid, Spain: IEEE.

Cavalcanti, Y. C., da Silveira Mota, P. A., de Almeida, E. S., Lucrédio, D., da Cunha, CEA., & de Lemos Meira, S. R. (2010b). One step more to understand the bug report duplication problem. In *XXIV Simpósio Brasileiro de Engenharia de software* (*SBES'2010*), Salvador, Brazil.

da Cunha, C. E. A., Cavalcanti, Y. C., da Mota Silveira Neto, P. A., de Almeida, E. S., & de Lemos Meira, S. R. (2010). A visual bug report analysis and search tool. In *Proceedings of the 22nd international conference on software engineering and knowledge engineering* (*SEKE'2010*) (pp. 742–747), San Franciso, CA.

D'Ambros, M., & Lanza, M. (2006). Software bugs and evolution: A visual approach to uncover their relationship. In *Proceedings of the 10th European conference on software maintenance and reengineering* (*CSMR'06*) (pp. 229–238). New York: IEEE Press. doi:10.1109/CSMR.2006.51.

Efron, B., & Tibshirani, R. J. (1993). *An introduction to the bootstrap*. New York, NY: Chapman and Hall/CRC.

Feldman, R., & Sanger, J. (2007). *The text mining handbook: Advanced approaches in analyzing unstructured data*. Cambridge: Cambridge University Press.

Fischer, M., Pinzger, M., & Gall, H. (2003a). Analyzing and relating bug report data for feature tracking. In *Proceednings of the 10th working conference on reverse engineering* (*WCRE'03*) (pp. 90–99). Washington, DC: IEEE Press.

Fischer, M., Pinzger, M., & Gall, H. (2003b). Populating a release history database from version control and bug tracking systems. In *Proceedings of the 19th international conference on software maintenance* (*ICSM'03*) (pp. 23–32). New York: IEEE Press. doi:10.1109/ICSM.2003.1235403.

Hiew, L. (2006). *Assisted detection of duplicate bug reports*. Master's thesis, The University of British Columbia.

Jalbert, N., & Weimer, W. (2008). Automated duplicate detection for bug tracking systems. In *Proceedings of the 38th annual IEEE/IFIP international conference on dependable systems and networks* (*DSN'08*) (pp. 52–61). New York: IEEE Press.

Jeong, G., Kim, S., & Zimmermann, T. (2009). Improving bug triage with bug tossing graphs. In *Proceedings of the 7th joint meeting of the European software engineering confernce and the ACM SIGSOFT symposium on the foundations of software engineering* (*ESEC/FSE'2009*).

Johnson, J. N., Dubois, P. F. (2003). Issue tracking. *Computers in Science and Engineering, 5*(6), 71–77.

Ko, A. J., Myers, B. A., & Chau, D. H. (2006). A linguistic analysis of how people describe software problems. In *Proceedings of the visual languages and human-centric computing* (*VLHCC'06*) (pp. 127–134). Washington, DC: IEEE Press. doi:10.1109/VLHCC.2006.3.

Koponen, T., Lintula, H. (2006). Are the changes induced by the defect reports in the open source software maintenance? In H. R. Arabnia, & H. Reza (Eds.), *Procedings of the 2006 international confernce on software engineering research* (*SERP'06*) (pp. 429–435). Nevada, USA: CSREA Press.

Lancaster, F. W. (1986). *Vocabulary control for information retrieval* (2nd ed.). AL, USA: Information Resources Press.

Podgurski, A., Leon, D., Francis, P., Masri, W., Minch, M., Sun, J., & Wang, B. (2003). Automated support for classifying software failure reports. In *Proceedings of the 25th international conference on software engineering* (*ICSE'03*) (pp. 465–475). Washington, DC: IEEE Press. doi:10.1109/ICSE.2003.1201224.

Runeson, P., Alexandersson, M., & Nyholm, O. (2007). Detection of duplicate defect reports using natural language processing. In *Proceedings of the 29th international conference on software engineering* (*ICSE'07*) (pp. 499–510). New York: IEEE Press. doi:10.1109/ICSE.2007.32.

Sandusky, R. J., Gasser, L., & Ripoche, G. (2004). Bug report networks: Varieties, strategies, and impacts in a f/oss development community. In *Proceedings of the 1st international workshop on mining software repositories* (*MSR'04*) (pp. 80–84). Waterloo: University of Waterloo.

Serrano, N., Ciordia, I. (2005). Bugzilla, itracker, and other bug trackers. *IEEE Software*, *22*(2), 11–13.

Sommerville, I. (2007). *Software engineering*, (8th ed.). New York: Addison Wesley.

Song, Q., Shepperd, M. J., Cartwright, M., & Mair, C. (2006). Software defect association mining and defect correction effort prediction. *IEEE Transactions on Software Engineering*, *32*(2), 69–82. doi:10.1109/TSE.2006.1599417.

Wang, X., Zhang, L., Xie, T., Anvik, J., & Sun, J. (2008). An approach to detecting duplicate bug reports using natural language and execution information. In *Proceedings of the 13th international conference on software engineering* (*ICSE'08*) (pp. 461–470). New York: ACM Press. doi:10.1145/1368088.1368151.

Weiss, C., Premraj, R., Zimmermann, T., & Zeller, A. (2007). How long will it take to fix this bug? In *Proceedings of the fourth international workshop on mining software repositories* (*MSR'07*) (pp. 20–26). New York: IEEE Press. doi:10.1109/MSR.2007.13.

Wohlin, C., Runeson, P., Martin Höst, M. C. O., Regnell, B., & Wesslén, A. (2000). *Experimentation in software engineering: An introduction the Kluwer internationational series in software engineering*. MA, USA: Kluwer Academic Publishers.

Author Biographies

Yguaratã Cerqueira Cavalcanti is Bachelor of Computer Science from Universidade Federal de Alagoas (2007), M.Sc. of Computer Science from Universidade Federal de Pernambuco (2009) and he is currently a Ph.D. student of Computer Science in Universidade Federal de Pernambuco and work as system development analyst at Serviço Federal de Processamento de Dados (SERPRO). Has experience in Software Engineering, acting on the following subjects: open source development, web applications development, mobile applications development, component based development and software configuration management and evolution. Contact him at yguarata@gmail.com.

Paulo Anselmo da Mota Silveira Neto has a Bachelor of Computer Science degree from Catholic University of Pernambuco (UNICAP), Specialist in Software Engineering from University of Pernambuco (UPE) and a Master of Science degree in Computer Science (Software Engineering) from Federal University of Pernambuco (UFPE). Nowadays, he is member of the Reuse in Software Engineering (RiSE) Group, which has executed research regarding to Software Product Lines (SPL) Testing, SPL Architecture Evaluation, Test Selection Techniques and Regression Testing. He is also participating on important research projects in Software Engineering area, as the National Institute of Science and Technology for Software Engineering (I.N.E.S.).

Daniel Lucrédio is an assistant professor at Federal University of São Carlos, Brazil, and senior member of the RiSE group (http://www.rise.com.br/). He got his computer engineer (2002) and M.Sc. in Computer Science (2005) degrees at the Federal University of São Carlos, Brazil, and his PhD (2009) at University of São Paulo, São Carlos, Brazil. Daniel has worked for 6 months (2007) at George Mason University (VA, USA), as a visiting scholar, and 12 weeks (2008) at Microsoft Research (WA, USA), as a Latin American MSR fellow, performing research in model-driven engineering. He has academic and industrial experience in computer science (software engineering) working mainly with software reuse, CASE, model-driven engineering, component search and component-based development.

Tassio Vale is a master student at Computer Science from Universidade Federal de Pernambuco (UFPE), currently part of the research group RiSE (Reuse in Software Engineering) in the study of Software Product Lines and Software Architecture. He has experience on issues related to software engineering, databases, and qualitative research.

Eduardo Santana de Almeida is an assistant professor at Federal University of Bahia and head of the Reuse in Software Engineering (RiSE) Labs. He has more than 100 papers published in the main conferences and journals related to Software Engineering and has chaired several national and international conferences and workshops. His research areas include: methods, processes, tools and metrics to develop reusable software. Contact him at esa@dcc.ufba.br.

Silvio Romero de Lemos Meira has a degree in Electronic Engineering from the Instituto Tecnológico de Aeronáutica (1977), Masters in Computer Science from Universidade Federal de Pernambuco (1981) and Ph.D. in Computer Science at University of Kent at Canterbury (1985). He is currently a professor at Universidade Federal de Pernambuco. He has experience in computer science, with emphasis on Software Engineering, working on the following topics: software reuse, information systems, open source, social networking, performance, and quality metrics in software engineering.