Characterizing Verification of Bug Fixes in Two Open Source IDEs

Rodrigo Souza and Christina Chavez
Software Engineering Labs
Department of Computer Science - IM
Federal University of Bahia (UFBA), Brazil
{*rodrigo,flach*}*@dcc.ufba.br*

***Abstract*—Data from bug repositories have been used to enable inquiries about software product and process quality. Unfortunately, such repositories often contain inaccurate, inconsistent, or missing data, which can originate misleading results.**

In this paper, we investigate how well data from bug repositories support the discovery of details about the software verification process in two open source projects, Eclipse and NetBeans. We have been able do identify quality assurance teams in NetBeans and to detect a well-defined verification phase in Eclipse.

A major challenge, however, was to identify the verification techniques used in the projects. Moreover, we found cases in which a large batch of bug fixes is simultaneously reported to be verified, although no software verification was actually done.

Such mass verifications, if not acknowledged, threatens analyses that rely on information about software verification reported on bug repositories. Therefore, we recommend that the exploratory analyses presented in this paper precede inferences based on reported verifications.

***Keywords*-mining software repositories; bug tracking systems; software verification; empirical study.**

I. Introduction

Bug repositories (or bug tracking systems) have for a long time been used in software projects to support coordination among stakeholders. Such systems record discussion and progress of software evolution activities, such as bug fixing and software verification. Hence, bug repositories are an opportunity to researchers who intend to investigate issues related to the quality of both the product and the process of a software development team.

However, mining bug repositories has its own risks. Previous research has identified problems of missing data (e.g., rationale, traceability links between reported bug fixes and source code changes) and inaccurate data (e.g., misclassification of bugs) [1].

In previous research [2], we tried to assess the impact of independent verification of bug fixes on software quality, by mining data from bug repositories. We relied on reported verifications tasks, as recorded in bug reports, and interpreted the recorded data according to the documentation for the specific bug tracking system used. As the partial results suggested that verification has no impact on software quality, we questioned the accuracy of the data about verification of bug fixes, and thus decided to investigate how verification is actually performed and reported on the projects that were analyzed, Eclipse and NetBeans.

Hence, in this paper, we investigate the following exploratory research questions regarding the software verification process in Eclipse and NetBeans:

- **How** is the verification performed: are there performed ad hoc tests, automated tests, code inspection?
- **Who** performs the verification: is there a dedicated team for QA?
- **When** is the verification performed: is it performed just after the fix, or is there a verification phase?

The next section contains some background information about bug tracking systems and software verification. In Section **??**, the data and methods used to investigate the research questions are presented. Then, in Section IV, the results are exposed and discussed. Finally, Section V presents some concluding remarks.

II. Background

Bug tracking systems allow users and developers of a software project to manage a list of bugs for the project. Usually, users and developers can report bugs, along with information such as steps to reproduce the bug, its severity and the operating system used. Developers choose bugs to fix and can report on the progress of the bug fixing activities, ask for clarification, discuss causes for the bug etc.

In this research, we focus on Bugzilla, an open source bug tracking system used by notable software projects such as Eclipse, Mozilla, Linux Kernel, Open Office, Apache, and companies such as NASA and Facebook[1]. The general concepts from Bugzilla should apply to most other bug tracking systems.

One important feature of a bug that is recorded on bug tracking systems is its status. The status records the progress of the bug fixing activity, and, as such, provides data to software engineering studies. Figure 1 shows each status that can be recorded in Bugzilla, along with typical transitions between status values, i.e., the workflow.

In the simplest cases, a bug is created and receive the status `UNCONFIRMED` (if it was created by a regular user)

[1]Complete list available at http://www.bugzilla.org/installation-list/.

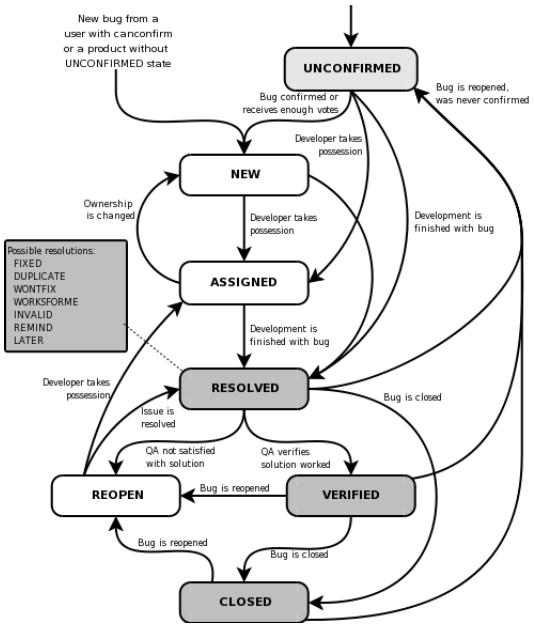

Figure 1. Workflow for Bugzilla. Source: http://www.bugzilla.org/docs/2.18/html/lifecycle.html.

or NEW (if it was created by a developer). Next, it is ASSIGNED to a developer, and then it is RESOLVED, possibly by fixing it with a patch on the source code. The solution is then VERIFIED by someone in the quality assurance team. If it passes the quality requirements, it is CLOSED when the next release of the software is released. If it does not pass the quality requirements or if the solution only partially fixes the problem, the bug is REOPENED.

In order to understand the bug fixing process of a software team, however, it should be possible to know what activities are performed upon change of status. What do developers do before marking a bug as VERIFIED? Bugzilla documentation states that, when a bug is VERIFIED it means that "QA [quality assurance team] has looked at the bug and the resolution and agrees that the appropriate resolution has been taken"[2]. Again, the definition is broad, although it assumes the existence of a QA team.

The developer documentation for both Eclipse and NetBeans are unclear about how the verification of a bug fix is performed. So, what forms of software verification are they used in the verification process of these projects?

Software verification techniques are classified in static and dynamic [3]. Static techniques include source code inspection, automated static analysis, and formal verification. Dynamic techniques, or testing, involve executing the software system under certain conditions and comparing its actual behavior with the intended behavior. Testing can be done in an improvised way (ad hoc testing), or can be compiled in a list of test cases, which can then be automated (e.g., as unit tests).

While mining bug repositories, one cannot assume that the VERIFIED status comprises all forms of software verification. Also, one cannot rely on the information provided by the bug tracking system or the developer documentation for a project, since it can be too generic to be useful. Such information can only be assessed by taking a closer look on actual bug reports.

III. Method

In order to answer the research questions—when and how bug fixes are verified, and who verifies them—, a three-part method was used:

1) **Data extraction**: we have obtained publicly available raw data from the Bugzilla repositories of two popular integrated development environments, Eclipse and NetBeans.
2) **Data sampling**: for each project, two representative subprojects were chosen for analysis.
3) **Data analysis**: for each research question, a distinct analysis was required. The analyses will be further described.

A. Data Extraction

In order to perform the desired analyses, we needed access to the data recorded by Bugzilla for a specific project, including status changes and comments. We have found such data for two projects—Eclipse and NetBeans—from the domain of integrated development environments. The data were made available as part of the Mining Software Repositories 2011 Challenge[3] in the form of MySQL database dumps. The files contain all data from the respective databases, except for developer profiles, that were omitted for privacy reasons.

Eclipse development began in late 1998 with IBM[4]. It was licensed as open source in November, 2001. The available data set contains 316,911 bug reports for its 155 subprojects, from October, 2001 to June, 2010.

NetBeans[5] started as a student project in 1996. It was bought by Sun Microsystems in October, 1999, and it was open sourced in March, 2000. The data set contains 185,578 bug reports for its 39 subprojects, from June, 1998 to June, 2010.

B. Data Sampling

Four subprojects were chosen for further analysis: Eclipse/Platform, Eclipse/EMF, Netbeans/Platform, and Netbeans/VersionControl. The Platform subprojects are the main subprojects for the respective IDEs, so they are both important and representative of each projects' philosophy.

[2]https://landfill.bugzilla.org/bugzilla-3.6-branch/page.cgi?id=fields.html

[3]http://www.msrconf.org/msr-challenge.html

[4]http://www.ibm.com/developerworks/rational/library/nov05/cernosek/

[5]http://netbeans.org/about/history.html

The other two subprojects were chosen at random, restricted to subprojects in which the proportion of verified bugs was greater than the proportion observed in the respective Platform subprojects. The reason is to avoid selecting projects in which bugs are seldom marked as VERIFIED. The following proportions of VERIFIED bugs per project were observed: Eclipse/Platform: 16.0%; Eclipse/EMF: 48.4%; NetBeans/Platform: 21.4%; NetBeans/VersionControl: 29.7%.

C. Analysis: How Are Bugs Verified?

In order to discover the verification techniques used by the subprojects, we have looked at the comments written by developers when they mark a bug as VERIFIED (meaning that the fix was accepted) or REOPENED (meaning that the fix was rejected). First, an informal analysis was performed, by reading a sample of such comments, looking for textual patterns that indicate that a particular technique was used. Then, based on the informal analysis and on previous knowledge, a regular expression was built for each technique, that should match comments in which there is evidence that that particular technique was used. The regular expressions were refined by looking at the results matched by them and modifying them to avoid false positives.

The regular expressions can be downloaded at http://.... For the sake of simplicity, the following list describes only some substrings matched by the regular expressions for each technique:

- automated testing: testcase, test case, unit test, automated test, test result, run the test, test run;
- source code inspection: inspecting, inspection, look at the code, looking at the source;
- ad hoc testing: verified by running, verified by ...ing;
- automated static analysis: findbugs, checkstyle, static code, static analys;
- formal testing: formal test, formal verif.

During the informal analysis, it was observed that ad hoc testing is reported in many distinct ways that are difficult to capture by means of regular expressions. Because of this, the regular expression for is expected to match only a few comments.

D. Analysis: When Are Bugs Verified?

In order to determine if there is a well-defined verification phase for the subprojects, we have selected all reported verifications (i.e., status changes to VERIFIED) over the lifetime of each subproject. Then, we have plotted, for each day in the interval, the accumulated number of verifications reported since the first day available in the data. The curve is monotonically increasing, with steeper ascents representing periods of intense verification activity.

Also, we have obtained the release dates for multiple versions of Eclipse and NetBeans. The information was obtained in the respective websites. In cases in which older information was not available, archived versions of the web pages were accessed via the website www.archive.org.

If a subproject presents a well-defined verification phase, it is expected that the verification activity is more intense a few days before a release. Such pattern can be identified by visual inspection of the graph, by looking for steeper ascents in the verification curve preceding the release dates.

E. Analysis: Who Verifies Bugs?

In order to determine whether there is a quality assurance (QA) team, we have counted how many times each developer has marked a bug as FIXED or VERIFIED. We considered that a developer is part of a QA team if s/he verified at least 10 times more than s/he fixed bugs. Also, we have computed the proportion of verifications that was performed by the discovered QA team. It is expected that, if the discovered set of developers is actually a QA team, they should be responsible for the majority of the verifications.

IV. Results and Discussion

Each of the three analysis was performed on Eclipse/Platform, Eclipse/EMF, NetBeans/Platform, and NetBeans/VersionControl. The results are presented next.

A. How Are Bugs Verified?

Most verification comments do not provide any clue about the verification technique used. Most often, developers just state that the bug fix was verified, sometimes informing the build used to verify the fix.

Table I shows, for each subproject, the number of comments associated with a bug being marked as VERIFIED or REOPENED that refers to a particular verification technique. No references to formal verification or static analysis were found; therefore, these techniques are not shown on the table.

Table I
VERIFICATION TECHNIQUES FOR ALL FOUR SUBPROJECTS.

Project	Testing	Inspection	Ad Hoc
Eclipse/Platform	260 (1.1%)	524 (2.2%)	74 (0.3%)
Eclipse/EMF	27 (0.7%)	2 (0.1%)	0 (0.0%)
NetBeans/Platform	94 (0.6%)	5 (0.0%)	0 (0.0%)
NetBeans/VersionControl	4 (0.1%)	1 (0.0%)	0 (0.0%)

In Eclipse/Platform, less than 4% of the comments referenced the verification technique that was used; for the other projects, less than 1% of the comments contained such references. Such low proportion reveals how difficult it is to infer the verification technique that was applied using only data from bug repositories. This is probably tacit knowledge that can only be extracted by asking the developers themselves.

In all subprojects, comments suggest the use of automated testing and code inspection in the verification process, the former technique being more frequently referenced (except in Eclipse/Platform, in which code inspection is cited more

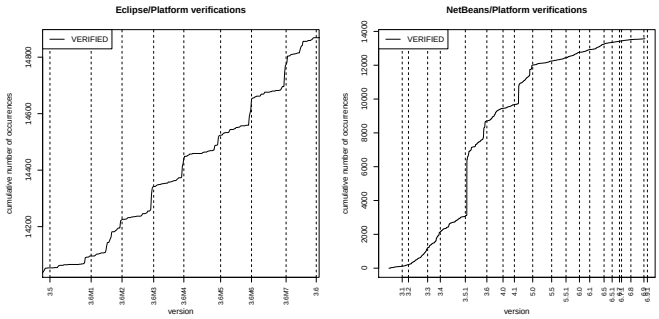

Figure 2. Number of verifications over time.

often). Evidences of ad hoc testing were found only in Eclipse/Platform, because verification comments tend to be more structured in this project, but we believe that ad hoc testing is much more common.

While sorting through comments, we have found interesting information regarding the software verification process. In Eclipse/Platform, the developer who fixes a bug often someone else to verify the fix, by asking "please verify, [developer name]". If the bug is reopened, then the fixer and the verifier exchange roles. This behavior illustrates a structured bug fixing/verification process.

In Eclipse/EMF, we found that marking a bug as `VERIFIED` does not mean that the bug fix was verified. Instead, it means that the fix was made available in a build of the software that is published in the subproject's website. This reinterpretation of the Bugzilla workflow was decided in a meeting[6].

Conclusion: it's hard to determine the technique used by just looking at the data. Eclipse/Platform documents it better, sometimes.

B. When Are Bugs Verified?

Figure 2 shows plots of total accumulated number verifications over time for Eclipse/Platform (left) and NetBeans/Platform (right). Releases are plotted as dashed vertical lines.

For Eclipse/Platform, the graph shows the period between releases 3.5 and 3.6, including milestone releases every 6 to 7 weeks. It is clear from the graph that the verification activity is intensified in a few days preceding a milestone, represented in the graph by steeper ascents before the vertical lines. This pattern is an indicator of a verification phase. No such pattern was found in the other subprojects by analyzing their graphs (not shown here for brevity).

The graph for NetBeans/Platform (Figure 2, right side) shows the entire project history. Although there are steeper ascents, they are different because they do not precede release dates. Also, at a closer look, they represent thousands of verifications performed in a few minutes by the same developer, with the same comment. The same pattern was found in Eclipse/EMF (not shown).

Of course, no developer can verify so many fixes in so little time. The explanation, supported by the comments, is that such mass verifications represent some cleanup of the bug repository, by marking old bugs as `VERIFIED`—with no verification being actually performed. Researchers should take extra care with mass verifications, as they contain a large amount of bugs and, thus, are likely to bias the results of analyses.

[6]http://wiki.eclipse.org/Modeling_PMC_Meeting,_2007-10-16

Table II
DISCOVERED QA TEAM FOR ALL FOUR SUBPROJECTS.

Project	QA team size	% of verifications by QA team
Eclipse/Platform	4 (2.4%)	1.1%
Eclipse/EMF	0 (0.0%)	0.0%
NetBeans/Platform	5 (21.7%)	93.3%
NetBeans/VersionControl	5 (20.8%)	93.2%

C. Who Verifies Bugs?

Table II presents, for each subproject, the number of developers attributed to a QA team (i.e., developers who perform verifications at least 10 times more often than contribute bug fixes), and the proportion of bug verifications they account for.

In NetBeans (both Platform and VersionControl) it is possible to infer the existence of a QA team, composed by approximately 20% of the developers, that performs about 90% of all verifications. In the Eclipse subprojects, there is no evidence of a dedicated QA team. In Eclipse/EMF no developer focus specifically on verification. In Eclipse/Platform, 2.4% of the developers focus on verification tasks, contributing only to 1.1% of the verifications.

V. CONCLUSION

By analyzing four subprojects (two from Eclipse, two from NetBeans), we have found, using only data from bug repositories, subprojects with and without QA teams, with and without a well-defined verification phase. We also have found evidence of the application of automated testing and source code inspection, although such evidence was less reliable. Also, we have found cases in which marking a bug as `VERIFIED` does not imply that any kind of software verification was actually performed.

With the knowledge obtained from this exploratory research, we aim to improve and extend our previous work on the impact of independent verification on software quality. We can investigate, for example, whether verification performed by QA team is better than verification performed by other developers.

Researchers should be aware that information about verification techniques are not common in bug repositories, and that reported verification does not always correspond to

actual verification. Some exploration of the data is important to avoid such pitfalls.

ACKNOWLEDGMENT

The first author is supported by FAPESB under grant BOL0119/2010.

REFERENCES

[1] J. Aranda and G. Venolia, "The secret life of bugs: Going past the errors and omissions in software repositories," in *Proceedings of the 31st International Conference on Software Engineering*.

[2] R. Souza and C. Chavez, "Impact of the four eyes principle on bug reopening: the case of eclipse," Federal University of Bahia, Tech. Rep., Aug 2011.

[3] I. Sommerville, *Software engineering (5th ed.)*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1995.