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 the reported status of bug reports from the projects Eclipse and Netbeans, in special the VERIFIED status, corresponding to the verification that a bug fix is appropriate. We show that the VERIFIED status is used in a variety of distinct ways, not always consistent with the traditional definition of software verification. Future research should take into account the multiple contexts in which the VERIFIED status is used when deriving conclusions based on the data from bug reports.**

***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 corrective, perfective, and ... changes [**?**]. Hence, bug repositories are an opportunity to researchers who intend to investigate issues related to the quality of the product and of the process of software development team.

The information contained in bug repositories has been used in order to predict fault-proneness, to unveil developers' roles from data, to characterize transfer of work in software projects, and so on. Also, it has been used to investigate beliefs in software engineering related to software quality, such as the impact of developer turnover and of collective ownership on the occurrence of bugs.

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], [2], [3], [4].

In previous research [cite report], 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, ad thus decided to investigate how verification is actually performed and reported on specific software projects.

In particular, we are interested in answer the following exploratory research questions for a limited set of projects:

- **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?

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) 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.

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

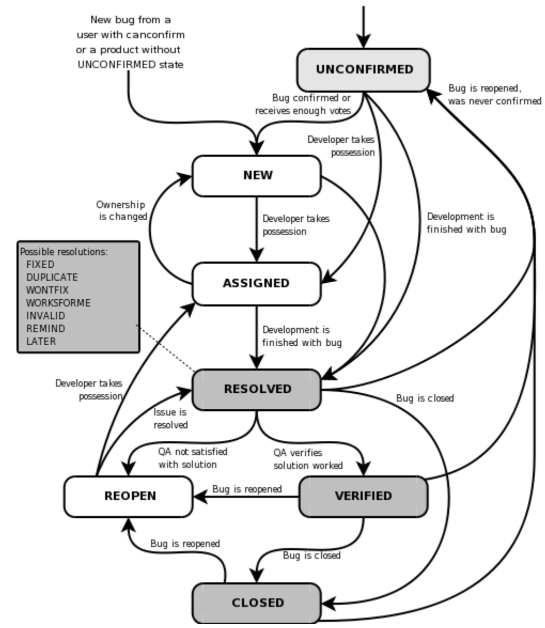

Figure 1. Workflow for Bugzilla. Source: http://www.bugzilla.org/docs/2.18/html/lifecycle.html.

If it does not pass the quality requirements or if the solution only partially fixes the problem, the bug is REOPENed.

TODO: Roles: reporter, fixer, verifier, reopener, closer.

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. For example, what steps are made before changing the status of a bug to RESOLVED (FIXED)? Is the patch applied to the version control system? Is it presented as diff file attached to the bug report? Is there a build available to end users containing the patch (e.g., a nightly build)? The documentation for Bugzilla does not prescribe specific activities for marking a bug as RESOLVED, relegating instead this decision to each software project.

Similarly, 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 [cite Sommervile]. 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 close look on actual bug reports.

III. Method

TODO: the method has changed. Please ignore this section.

In order to answer our exploratory research questions, we followed the following method:

1) **Data extraction**: we have obtained publicly available raw data from the Bugzilla repositories of two popular integrated development environments, Eclipse and Bugzilla.
2) **Data sampling**: we have chosen two subprojects. After that, we sampled bugs with interesting characteristics.
3) **Data analysis**: we have read the full bug reported for the sampled bugs. Also, we have performed quantitative analysis on all bugs.
4) ...

A. Available Data

In order to perform the desired analysis, we needed access to all data recorded by Bugzilla for a specific project. 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 tables from the respective databases, except for the tables containing information about developers.

Eclipse development began in late 1998 with IBM[4]. It was licensed as open source in November, 2001. In 2004, IBM created the Eclipse Foundation in order to reduce the perception of Eclipse as a IBM-controlled project.

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.

Both Eclipse and NetBeans are composed of many subprojects. The data comprises all bug reports from all Eclipse subprojects from October, 2001 to June, 2010, and from all NetBeans subprojects from June, 1998 to June, 2010.

[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

There are 316,911 bug reports for 155 subprojects in Eclipse, and 185,578 bug reports for 39 subprojects in NetBeans.

B. Sampling of Bug Reports

In order to obtain insight about the research questions, we read a sample of bug reports from both projects. For each project, two subprojects were chosen randomly among the projects with most verified bugs, resulting in the following subprojects: Eclipse/Platform, Eclipse/EMF, Netbeans/platform, Netbeans/versioncontrol. We have sampled 20 bug reports for each subproject, totalizing 80 bug reports.

TODO: this sampling was made just for the preliminary, qualitative analysis. Maybe we should describe preliminary analysis as an unstructured inquiry (instead of detailing the methods for it).

IV. Results and Discussion

A. Preliminary Analysis

Some insights were gained in this step, partially answering the who, when, and how questions:

- how In some projects, the status VERIFIED does not correspond to software verification. In Eclipse/EMF, for example, it means that the fix is available in a build that is promoted on the project website[6]. Further evidence: bug 254489 “Please do not change the state of the bugs since the status verified fixed should be only there when we publish a new build. :-)”
- (who) In the NetBeans subprojects that were analyzed, it was common that the reporter also verified the fix. In some bugs, this was necessary since the developer was not able to reproduce the bug.
- (who, when) Often, the fixer verifies the bug just after marking it as fixed.
- (when) Sometimes, several old bugs are marked as VERIFIED at the same time.
- (how?) In both Eclipse and NetBeans, it is common that the fixer requests someone else to verify the fix (“Please verify, Tom”).
- (how) In Eclipse/EMF, bug 249436, one developer rewrites the patch submitted by other developer.
- (how) In Eclipse/EMF, bug 269789, a test case is used to assess the proposed fix.
- (how) Eclipse/Platform, bug 210533: “Verified by code inspection”.

TODO: : “Verified in/with [build]”, “Verified by/through [verification technique]”.

B. RQ1: How Are Bugs Verified?

We analyzed cases in which the bug is fixed, the fixer asks someone else to verify the fix, and then the bug is reopened.

Findings: ...

[6]http://wiki.eclipse.org/Modeling_PMC_Meeting,_2007-10-16 (between releases 3.3.1 and 3.3.1.1)

C. RQ2: When Are Bugs Verified?

In order to answer whether there is a verification phase for the projects analyzed, we plotted the cumulative number of verifications along the lifetime of each project, as shown in Figure XXX, in red. The vertical lines represent release dates for each version. As a basis for comparison, the number of bug fixes is also plotted, in black.

We found evidence of a verification phase for Eclipse/Platform. In this project, verifications appear to be performed in bursts, with periods of intense activity separated by about 6 to 7 weeks of low or no activity. By taking a closer look in the period between the releases 3.5 and 3.6, it is clear that the periods of intense verification activity occur just before the milestones leading to 3.6.

No such pattern was found for Netbeans’ subprojects.

The first thing that comes to attention are “cliff walls”, points in time when a large amount of bugs are verified, represented by steep, almost vertical portions of the plotted line. They appear in Eclipse/EMF and Netbeans/Platform. In Netbeans/Platform, all verifications in the cliff wall preceding version 5.5 share the same message: “Verification of old issues”. For Eclipse/EMF, the message is “Move to verified as per bug 206558”, which refers to the task of changing bugs marked as FIXED to VERIFIED.

Cliff walls represent a change in the development process, and the mass verification is needed in order to leverage useful features from Bugzilla (e.g., looking for verified bugs in order to aid the writing of release notes).

Because of cliff walls, extra care must be taken before doing any analysis that relies on the VERIFIED status. In such situations, the VERIFIED status is applied blindly, with no software verification being actually performed. Because cliff walls contain a large amount of bugs, they are likely to bias the results of analyses.

D. RQ3: Who Verifies Bugs?

We counted the number of times each developer verified or fixed a bug. We consider that, if a developer is primarily concerned with quality assurance, then s/he verifies much more bugs that s/he fixes.

We defined ratio as ...

The verified/fixed ratio for the four projects is shown on Table XXX.

```
*** eclipse-platform
Verifiers: 4 (2.4%)
  1.1% of verifications
Fixers: 30 (17.9%)
Total developers: 168

*** eclipse-emf
Verifiers: 0 (0.0%)
  0.0% of verifications
Fixers: 4 (30.8%)
```

```
Total developers: 13

*** netbeans-platform
Verifiers: 5 (21.7%)
  93.3% of verifications
Fixers: 6 (26.1%)
Total developers: 23

*** netbeans-versioncontrol
Verifiers: 5 (20.8%)
  93.2% of verifications
Fixers: 6 (25.0%)
Total developers: 24
```

In NetBeans 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, only 2% of the developers focus on verification tasks, however, they contribute only to 1% of the verifications.

V. Conclusion

The VERIFIED status is used in a variety of distinct ways, not always consistent with the traditional definition of software verification ...

With the knowledge obtained from this exploratory research, we can improve and extend our previous work on the impact of independent verification on software quality. Examples of refined questions:

- Does independent verification improve software quality when it is performed by developers from the QA team?
- Does independent verification improve software quality when it is performed as an separate phase of the software release cycle?
- Does independent verification improve software quality when a particular verification technique (e.g., inspections, unit testing) is used?

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*, ser. ICSE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 298–308. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2009.5070530

[2] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu, "Fair and balanced?: bias in bug-fix datasets," in *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ser. ESEC/FSE '09. New York, NY, USA: ACM, 2009, pp. 121–130. [Online]. Available: http://doi.acm.org/10.1145/1595696.1595716

[3] T. H. D. Nguyen, B. Adams, and A. E. Hassan, "A case study of bias in bug-fix datasets," in *WCRE*, G. Antoniol, M. Pinzger, and E. J. Chikofsky, Eds. IEEE Computer Society, 2010, pp. 259–268.

[4] K. Ayari, P. Meshkinfam, G. Antoniol, and M. Di Penta, "Threats on building models from cvs and bugzilla repositories: the mozilla case study," in *Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*, ser. CASCON '07. New York, NY, USA: ACM, 2007, pp. 215–228. [Online]. Available: http://doi.acm.org/10.1145/1321211.1321234