Verification of Bug Fixes: What Does it Really Mean?

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 qualitatively 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 verification. Future research should take the multiple meanings of VERIFIED into account when deriving conclusions based on the status field of bug reports.**

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

I. Introduction

Bug repositories (or bug tracking systems) have since 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 a software development team, which can give insights about the nature of software development itself.

The information contained in bug repositories have 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].

TODO: why the VERIFIED status is so important?

In summary, for researchers who intend to use data from bug tracking systems, it is not sufficient to understand the data fields in a bug report the way they are documented in the system's documentation. The research should also investigate whether the data fields are used often, whether they are used accordingly to the documentation, and even if the data fields are used consistently across the developers.

In this paper, we investigate the status of a bug, a data field present in most bug tracking systems that reports the progress of the bug solving activity. In particular, we focus on the activity of verification of a fix. We are interested in answering the following exploratory research questions:

- What activity is actually performed when a bug fix is marked as verified?
- When is the verification performed (i.e., just after the fix, or is there a specific period for the verification of fixed bugs?)
- How is the distribution of verification across developers?
- Is there a dedicated team for QA?

For research question xx, qualitative of a small sample. For research question xx, all bugs were considered.

II. Background

A. Bug Tracking Systems

Bug workflow. Bugzilla. [2]

Bug tracking systems allow users and developers of a software system to manage a list of bugs for the system. 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]. However, 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.

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

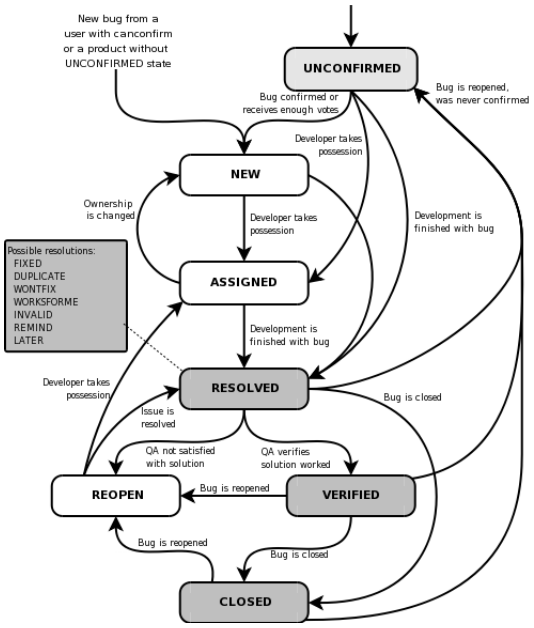

Figure 1. Workflow for Bugzilla. Source: http://www.bugzilla.org/docs/2.18/html/lifecycle.html.

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. 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. For example, what steps are made before changing the status of a bug to RESOLVED (fixed)? Has the patch been applied to the version control system or is it presented as diff file attached to the bug report? Has the patched source code been built and made available for end users (e.g., as 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"[https://landfill.bugzilla.org/bugzilla-3.6-branch/page.cgi?id=fields.html]. Again, the definition is loose, although it assumes the exi

IV. Data Extraction

We have had access to Eclipse Bugzilla database from October, 2001 to June, 2010, in MySQL format. The data was made available as part of the Mining Software Repositories 2011 Challenge[2].

The database includes data from 155 Eclipse subprojects. However, we have discarded subprojects in which no bug was ever reopened, and subprojects with less than 50 reported bugs.

For the remaining 34 subprojects, we have selected only bugs that were verified until December 2009. The rationale is that if, within six months, a verified bug was not reopened, it probably will not be reopened after that and it is, thus, fully fixed.

A. The Eclipse Project

History.
Recommendations about Bugzilla usage. Four-eyes principle.

The Eclipse Project was created in 2001 by IBM and released into open source in 2004[3], with focus on the creation of a extensible platform that supports the software development lifecycle. Today, it is maintained by its open source community, as well as by IBM employees and people from other organizations.

The Eclipse Project consists of many subprojects, including the Platform project and dozens of plugins. Although each subproject is maintained by a distinct team and is relatively autonomous, all of them should follow a general, documented process.

All subprojects use Bugzilla for bug reporting, and there are clear rules regarding the process of change in the software[4]. Here is an excerpt of a document in the Eclipse web site that explains the bug fixing process[5] (emphasis added):

> When a new bug is entered it begins life with a Status of either **Unconfirmed** for normal users or **New** for users with commit privileges. The bug is typically assigned to the component owner. The component owner will usually use a query of Status = Unconfirmed or New and Assigned to = me to browse what is essentially the component's inbox. She or he will assign bug reports to developers. Email will be generated to the developer (this part is configurable). (Note that the act of reassigning a bug to somebody changes its status to New). The developer may change unconfirmed bugs to new.
> The assigned developer will accept the bug which will change its status to **Assigned**. After working on the bug the developer will mark the bug as **Resolved** and will select a resolution (Fixed, Invalid, Wontfix, Worksforme).

[2]http://www.msrconf.org/msr-challenge.html
[3]http://www.eclipse.org/org/
[4]http://wiki.eclipse.org/Development_Resources/HOWTO/Bugzilla_Use
[5]http://wiki.eclipse.org/Bug_Reporting_FAQ

> Once a bug is resolved there are still a few states it can transition to. How you choose to use these states will be up to individual teams. In our example, when a test pass comes up a component owner can build a query that searched for resolved bugs. After testing that the fix worked a resolved bug can be transitioned to **verified**, directly to **closed**, or in fact, **reopened**. By searching for bugs with a Status of verified and a resolution of fixed developers can come up with release notes. A verified/fixed bug can then be transitioned to closed. And yes, closed bugs can be reopened if need be. As an added bonus other bug status can be transitioned through verified to closed as well. This gives developers the opportunity to test worksforme claims.

Theres also a recommendation to follow the four eyes principle in the verification process (emphasis added):

> When a developer fixes the bug, the status is set to RESOLVED - FIXED, and it is assigned to another developer on the team to verify. **It is important that the verifier be a different person than the fixer** because the fixer is too close to the code and thus may not be as diligent at testing the corner cases.

Finally, there are additional guidelines regarding the workflow:

- All bugs should be verified before the next integration build.
- When a committer verifies a fix, the status is changed to VERIFIED.
- When the project does a major release, the VERIFIED bugs are changed to CLOSED.

B. The Netbeans Project

History.
Recommendations for developers.

V. Data Sampling

Scripts to classify interesting cases. Random sampling stratified by case.

VI. Analysis

grounded theory-style.

VII. Conclusion

VIII. Experimental Setup

```
Data extraction
Data processing:
- filtering (bugs up to december 20xx)
- classification script
Sampling:
- random sampling (for coding)
- entire data set
Analysis:
- grounded theory (open coding)
- latent Dirichlet allocation
- descriptive statistics
```

IX. Results

O que quer dizer a soluo de um bug foi verificada?

Como j foi comentado anteriormente, o Bugzilla permite que os relatrios de bugs sejam marcados como RESOLVED (resolvido) e, depois, como VERIFIED (verificado). Mas afinal, o que significa verificar a soluo de um bug? Que atividades ou checagens foram realizadas antes de se marcar o bug como VERIFIED?

A documentao do Bugzilla tem algo a dizer sobre isso o status VERIFIED:

QA [quality assurance team] has looked at the bug and the resolution and agrees that the appropriate resolution has been taken.

Essa definio sugere que a verificao deve ser feita por uma equipe especializada em controle de qualidade. Alm disso, o verificador deve concordar que a resoluo apropriada. A definio do Bugzilla para por a, mas eu consigo imaginar alguns critrios para considerar uma resoluo apropriada:

O bug no se manifesta aps a aplicao da soluo. A soluo no gera novos bugs. O cdigo da soluo segue as convenes de codificao e polticas do projeto. A soluo possui uma boa relao custo-benefcio isto , ou no existem solues melhores ou a soluo um paliativo por conta da alta complexidade de uma soluo melhor. Minha expectativa, portanto, que, antes de o bug ser marcado como VERIFICADO, ele passe pelas seguintes atividades:

Aplicao do patch com a soluo do bug na verso do software onde o bug foi encontrado. Execuo do software com o bug, seguindo os passos para reproduo do bug, e verificando que o bug no se manifesta Inspeo do cdigo-fonte. Deve ser verificada a aderłncia s convenes de codificao do projeto e a outras polticas (por exemplo, cada patch deve vir com testes de unidade). Execuo de testes de unidade (se houver). Deve se verificar que todos os testes passam. Bom, mas isso a teoria. Quais atividades so executadas na prtica, em projetos reais, antes de se marcar um bug como VERIFIED? Para responder a essa pergunta, analisei um total de 80 bugs, considerando os projetos Eclipse/EMF, Eclipse/Platform, Netbeans/versioncontrol e Netbeans/profiler. Eis alguns casos verificados nesses projetos:

O reporter (ou outro usurio que tambm enfrenta o problema) verificou que, em uma build do produto que inclui a soluo, o bug deixou de se manifestar. Aparentemente no h inspeo do cdigo-fonte, e nem uma equipe de QA. Situao frequente no Netbeans. Pode ser um procedimento necessrio quando os desenvolvedores no conseguem reproduzir o bug. A soluo est disponvel em um build que divulgado no site do projeto. Esse critrio usado pelo projeto Eclipse/EMF e foi decidido em reunio. O projeto usa o status FIXED para

bugs cuja soluo est no CVS. Bugs que j foram resolvidos h muito tempo so marcados como verificados, por se entender que, se nenhuma problema ocorreu at ento, isso significa que a soluo adequada. A mesma pessoa que resolve o bug o marca como VERIFIED logo depois. Aparentemente no existe verificao de fato nesse caso. O desenvolvedor que resolve o bug solicita que outro desenvolvedor verifique a soluo. Comum no Eclipse/Platform. No bug 249436 do Eclipse/EMF, um patch submetido por um desenvolvedor, mas outro desenvolvedor acaba reescrevendo o patch, fornecendo evidłncias de que o cdigo-fonte foi inspecionado. No bug 269789 do Eclipse/EMF, h evidłncias da elaborao e da execuo de testes de unidade. A anlise sugere que o status VERIFIED significa coisas diferentes em diferentes projetos. A execuo de uma verso corrigida do software parece ser o mtodo predominante de verificao. Em alguns casos, o status VERIFIED no representa verificao, e sim a indicao de que a soluo est disponvel para o usurio final. Evidłncias de inspeo de cdigo e execuo de testes de unidade s foram encontradas no projeto Eclipse/Platform.

X. Conclusion

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