Preventing Bug Reopening With Effective Software Verification

Rodrigo Souza Christina Chavez Roberto Bittencourt

July 7, 2013

1 Project Summary

Fixing bugs is an important software development task, often supported by bug tracking systems [5]. In a typical scenario, a bug report is created, then the bug is fixed, and finally the report is closed. Sometimes, however, someone discovers that the fix was incomplete or innapropriate. In this case, the bug report is reopened so it can be further investigated.

Reopened bugs take twice as much time to fix, on average [6], and involve the participation of 20 to 60% more developers [4]. Even worse is when developers only notice that a bug received an incomplete fix after the software was released. The result is that, because the buggy code was shipped, users' perception of the software quality is harmed.

To avoid post-release bug reopenings, in some projects bug fixes are verified. Verifying a bug fix means that someone independently checks if it is appropriate and complete. If it is not, the bug report is reopened right away, so it can be properly fixed before the next release.

Of course, verification is not perfect, and a bug can be reopened even after it has been considered appropriate during its verification. The verification, in this case, can be considered unsuccessful. But how to improve the bug fix verification success rate and, thus, reduce post-release bug reopening?

1.1 Objectives

We propose to investigate what are the dimensions of software verifications processes and how to reduce post-release bug reopening by tailoring the process along these dimensions. We are specifically interested in the verification of bug fixes (as opposed to the verification of new features).

In a previous work [8], we showed how to identify, by mining bug data, verification dimensions such as time (are bugs fixes verified at a constant rate or in bursts?), organizational structure (are verifications performed by specialized teams?), and technique (do verifications involve automated testing, code review or other techniques?). There

is little empirical evidence, however, on how these and other dimensions affect the success rate of verifications. In this project, we intend to expand on these dimensions and determine how they can help make the verification of bug fixes more effective.

1.2 Intellectual Merit

Recent works have tried to predict bug reopening using factors related to the bug, the bug fix, and the developers [7, 6, 12, 4, 1]. However, most findings relate bug reopening to factors that cannot be easily controlled and that, therefore, cannot be used to prevent reopening (for more details, see Section 2). In this project, we aim to determine how reopening can be made less frequent by improving the verification process.

Regarding the verification process, some studies try to empirically determine the relative effectiveness of distinct verification techniques, such as automated testing and code review [11]. Our approach differs in two aspects: methodology and scope.

Traditionally, empirical studies of verification techniques are based on experiments or quasi-experiments, and thus the results may fail to generalize or to apply to specific scenarios. In our project, we will leverage software development data, specifically bug reports, in order to find results that are relevant to a specific organization or development team.

Regarding scope, while most studies focus on a specific aspect of the software verification process—namely, the verification technique—, we will adopt a more holistic approach. We plan to study the influence of different dimensions of the software verification process—e.g., time and organizational structure—on its ability to detect problems with bug fixes.

1.3 Broader Impact

The expected tangible outcome of this project is a set of empirically supported guidelines to help improve the effectiveness of software verification processes and, thus, minimize the problems caused by bug reopening (see Section 1). The guidelines could potentially help improve both the productivity of a development team and the quality of the software produced.

The productivity would be increased by reducing the number of bugs that would be reopened after a failed verification, thus reducing the amount of rework. Besides, the quality would be improved by detecting problems with bug fixes before the next release, during verification.

2 Related Work

Despite its importance, only recently researchers have started to study bug reopening. Shihab and colleagues [7, 6] developed a decision tree model, based on features about the bug report, the bug fix, and human factors, to predict which bugs would be reopened. They found that the component the bug is in was among the top predictors of bug reopening.

In a partial replication of Shihab's work, Zimmermann and colleagues [12] found that bugs reported by users are more likely to be reopened than those discovered through code review or static analysis, supposedly because those reported by users are harder to reproduce and tend to be more complex. Other factors that favor reopening, according to the authors, include bug severity and geographical distribution of developers participating in the bug.

They also asked Microsoft engineers about the common causes for bug reopening. Responses included the difficulty to reproduce a bug, the misunderstanding of root causes, the lack of information in the initial report, the increase of the bug priority, incomplete fixes, and code integration problems.

Neither Shihab [7, 6] nor Zimmermann [12], though, made a distinction between bugs that were reopened after being fixed and those that were reopened after being only triaged (e.g., after the bug report was tagged as invalid or incomplete). In our opinion, these are different phenomena, with distinct causes and prevention approaches. In this proposal, we aim to investigate strictly the reopening of fixed bugs.

Other works deal specifically with bugs that were reopened after receiving a fix. Almossawi [1] analyzed 32 open source systems in the GNOME project and, using a logistic regression model, concluded that bugs located in code with high cyclomatic complexity are more likely to be reopened (after fixed). In our proposal, instead of investigating the influence of product metrics on reopening, we focus on the verification process.

Jongyindee and colleagues [3] found that bugs fixed by more experienced developers are less likely to be reopened. Although this result can be used to predict bug reopening, it cannot be used to prevent it, because it is generally unfeasible to allow only experienced developers to fix bugs. In this proposal, we are interested in ways of minimizing post-release reopening by improving a directly controllable aspect of software development: the software verification process.

3 Data and Schedule

This project relies on the following types of data (see CODEMINE's whitepaper [2], Figure 3): work item, organization, code review, process information, and, possibly, build and test (if they can be related to work items). The analysis will be centered on work items, and the other types of data will provide context (e.g., developer role at Microsoft or how bug was discovered [12]) to allow more refined inferences. The project would also benefit from interviews or surveys with Microsoft engineers.

We expect to conclude the project in 8 weeks, according to the following schedule: (1) learn about CODEMINE schema, API and tools; (2) identify verification practices at Microsoft through surveys and data analysis; (3) write and run scripts to clean up data; (4) write and run scripts to transform data; (5) write reports to characterize the data; (6) write and run analyses; (7) check findings with Microsoft engineers; (8) refine analyses.

4 People and Related Publications

One Ph.D. student and two professors are involved in the project. Below, a short bio for the people involved.

Rodrigo Rocha Gomes e Souza received a M.Sc. degree in Computer Science from the Federal University of Campina Grande (UFCG), Brazil, and is currently a Ph.D. student at the Federal University of Bahia (UFBA), Brazil. He has already done some work to support this proposal as part of his thesis, using data from open source projects, and contributes to the Data Analysis Community Portal[1]. His main research interests are in software evolution, mining software repositories, mobile applications, and computer games.

Christina von Flach Garcia Chavez received her Ph.D. in Computer Science at PUC-Rio, Brazil (2004). She is an associate professor at the Federal University of Bahia (UFBA), Brazil. She is the head of the Software Design and Evolution research group (aSide @ UFBA) and a member of the Software Engineering Labs (LES @ UFBA). She has co-authored over 50 refereed papers in journals, conferences and books. Her main research interests are in software architecture, software evolution, and software engineering education. Christina is a member of the ACM, IEEE, and SBC (Brazilian Computing Society).

Roberto Almeida Bittencourt received a Ph.D. degree in Computer Science from the Federal University of Campina Grande (UFCG), Brazil, that included a doctoral exchange in the University of British Columbia (UBC), Canada. He is currently an assistant professor at the State University of Feira de Santana (UEFS). He has published 20 papers in refereed conferences and workshops. His main research interests are in software evolution, mining software repositories, software engineering education, and collaborative systems.

We have published three papers that support this proposal. The first one was a characterization of the verification process of open source projects using data from bug reports. The results were presented at the 9th Working Conference on Mining Software Repositories (MSR 2012) [8].

This year we presented two papers [9, 10] at the Data Analysis Patterns in Software Engineering (DAPSE) workshop, organized by Microsoft Research and held in San Francisco, together with the 35th International Conference on Software Engineering (ICSE 2013). The papers include four patterns that help data scientists clean and transform bug data before it can be analyzed.

References

[1] A. Almossawi. Investigating the architectural drivers of defects in open-source software systems: an empirical study of defects and reopened defects in gnome, 2012.

[1] `http://dapse.unbox.org/`

[2] J. Czerwonka, N. Nagappan, W. Schulte, and B. Murphy. CODEMINE: Building a software analytics platform for collecting and analyzing engineering process data at Microsoft. Technical Report MSR-TR-2013-7, Microsoft Research, 2013.

[3] A. Jongyindee, M. Ohira, A. Ihara, and K.-i. Matsumoto. Good or bad committers? a case study of committers' cautiousness and the consequences on the bug fixing process in the Eclipse project. In *Proc. of the 2011 Joint Conf. of the 21st International Workshop on Softw. Measurement and the 6th International Conf. on Softw. Process and Product Measurement*, pages 116–125, Washington, DC, USA, 2011. IEEE Computer Society.

[4] J. Park, M. Kim, B. Ray, and D.-H. Bae. An empirical study of supplementary bug fixes. In *9th IEEE Working Conference on Mining Software Repositories*, pages 40–49, 2012.

[5] R. Patton. *Software Testing (2nd Edition)*. Sams, Indianapolis, IN, USA, 2005.

[6] E. Shihab. *An Exploration of Challenges Limiting Pragmatic Software Defect Prediction*. PhD thesis, School of Computing, Queen's University, Kingston, Ontario, Canada, 2012.

[7] E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K.-i. Matsumoto. Predicting re-opened bugs: A case study on the eclipse project. In *Proceedings of the 2010 17th Working Conference on Reverse Engineering*, WCRE '10, pages 249–258, Washington, DC, USA, 2010. IEEE Computer Society.

[8] R. Souza and C. Chavez. Characterizing verification of bug fixes in two open source IDEs. In *9th IEEE Working Conf. on Mining Software Repositories*, pages 70–73, 2012.

[9] R. Souza, C. Chavez, and R. Bittencourt. Patterns for cleaning up bug data. In *Proc. of the 1st Workshop on Data Analysis Patterns in Softw. Engineering*. IEEE, May 2013.

[10] R. Souza, C. Chavez, and R. Bittencourt. Patterns for extracting high level information from bug reports. In *Proc. of the 1st Workshop on Data Analysis Patterns in Softw. Engineering*. IEEE, May 2013.

[11] J. W. Wilkerson, J. F. Nunamaker, Jr., and R. Mercer. Comparing the defect reduction benefits of code inspection and test-driven development. *IEEE Trans. Softw. Eng.*, 38(3):547–560, May 2012.

[12] T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy. Characterizing and predicting which bugs get reopened. In *Proceedings of the 2012 International Conference on Software Engineering*, ICSE 2012, pages 1074–1083, Piscataway, NJ, USA, 2012. IEEE Press.