Categorizational Study on Commits For A Better Software Quality

Jincheng He
Department of Computer Science
University of Southern California
jinchenh@usc.edu

***Abstract*—Nowadays, it is common to see software being developed with support from online version control systems (VCS), such as Git. This makes it easier to track the development as well as to evaluate the quality of the software. At the same time, open source makes the projects more vulnerable [] because the developers can be from different organizations with different level of skills. Thus, open source software (OSS) quality investigation, control and improvements are one of the critical issues in the area of software engineering, and repository mining is one of the approaches to conduct them. Thus far, by repository mining, various researches have been conducted to assess the quality of software, but there is still space for investigations about analyzing how developer behaviors impact software quality and to extend the concept of software quality. In this research area, little work has been done in classification-based study on commits, focusing on types, commit message and code patterns, to analyze their relations with and impact on software quality. In this white paper, we list fields where and how further research can be done for the sake of a better software quality.**

***Index Terms*—Software Engineering, Software Maintenance, Software Quality, Open Source Software**

I. Introduction

Most industry-scale software are developed under iterative contributions from the project teams [], through ICSM, Agile, DevOps or other process models. In the iterations, version control systems, such as Git and SVN, play a critical role by enabling and facilitate the concurrent contributions from developers. Each revision, or commitment (referred as "commit" in the rest of the paper) contains diffs which are the lines developers change. These changes

Industry-scale software evolves through thousands of iterative code changes. Version control systems, such as Git and SVN, allow these changes to take place in parallel, with multiple developers contributing to a shared repository. Each contribution is captured in a commit – a collection of file diffs that indicate modified code. Developers author commits for different purposes. For example, a commit can edit the build configuration, fix a bug, or refactor existing code. It is common practice for developers to author atomic commits [1] that have a singular purpose. However, refactoring, added dependencies, minor fixes, and scope creep can cause the commit to grow beyond its intended task. Each commit includes a prescriptive message documenting the changes made [2], in practice with varying degrees of efficacy.

[1]https://www.freshconsulting.com/atomic-commits/

[2]https://git-scm.com/book/en/v2/Distributed-Git-Contributing-to-a-Project

A software revision created by a commit is expected to be compilable. However, uncompilability can occur due to careless development – failure to compile the software locally prior to pushing to the shared repository. It can also result from variations in build environments, incompatibility across overlapping changes made by multiple developers, or changes in upstream dependencies. The presence of compile errors inhibits bytecode and dynamic software analysis, as well as static analysis when the code is unparsable [1]. Previous studies [1]–[4] have shown that even in popular open-source projects maintained by major software organizations, build-breaking commits can occur.

Behnamghader et al. [5] explore the qualitative properties of uncompilable commits. Further insight into the types of commits that most frequently cause build errors can help to inform better development practices. Additionally, this correlation can be used as a part of future methods for predicting and preventing uncompilable commits. Analyzing the degradation of software quality over multiple uncompilable commits highlights the long-term negative impact of careless development, and the importance of fixing build errors immediately after they take place. Also, understanding the purposes of those uncompilable commits can result in preparing guidelines to avoid such degradation in the future.

Previous research has produced taxonomies for commit categorization [6]–[11]. While sharing some common category terminologies and definitions, these taxonomies cannot be directly adopted for understanding the purposes of commits. The taxonomies that integrate commit purpose have only been adapted to large commits [6], [11] which lack the specificity to classify smaller code changes [9].

In this paper, we analyze the differences in software quality metrics before and after sequences of uncompilable commits and compare them with the compilable sequences to study whether software quality changes differently over uncompilable periods. To gain a deeper insight into this, we explore the relation between commit size, purpose and uncompilability. We employ the commit purpose taxonomy from Hindle et al. [6], refine, standardize, and expand it to make it applicable for small commits. Our goal is to further understanding of the causes and effects of uncompilability in open-source systems, a significant problem that resists explanation by common software quality analysis tools. This work consists of the following contributions:

- A methodology for assessing software quality over sequences of uncompilable commits.
- Findings on the evolution of software quality over sequences of uncompilable commits.
- A definition for "independent changes" to help categorize commits.
- A taxonomy for categorizing commit purpose that can be used across commits of varying size.
- A dataset of 914 commits tagged using this taxonomy.
- Findings on the relationship between commit size and purpose.
- Findings on the incidence of commit purpose over uncompilable commits.

The remainder of the paper is organized as follows. Section **??** summarizes the related works that analyze compilability over commit history and the related works that categorize commits based on commit purpose. Section **??** discusses our research questions. Section **??** discusses our data collection process. Section **??** discusses our approach with regard to refining an existing taxonomy, to dealing with inconsistencies and ambiguities in the tagging process, and to studying software quality evolution over uncompilable periods. Section **??** discusses the results of our analysis with respect to the research questions. Section **??** focuses on the threats to validity of our analysis and Section **??** concludes the paper.

II. Commit Types

A. *A Refined Categorization*

B. *The Difference Between Categories with Respect to Software Quality*

III. Software Quality

A. *Tool-based Quality Metrics*

B. *Other Quality Aspects*

[1]

IV. Commit Message

A. *Initiatives of Commit Messages*

B. *Preprocessing Removal of Useless Information in Commit Messages*

C. *The Relation Between Commit Message and Software Quality*

V. Code Patterns

A. *Different Patterns Existing in Code.*

B. *Different Patterns May Have Different Impact on Software Quality*

VI. Conclusions

A. *Development Guidelines*

B. *A Novel Set of Software Quality Evaluation Metrics*

This direction mainly focuses on investigating the developer behaviors including how they contribute to the software project, especially to open source software where we can get enough metadata for analysis to evaluate how different behaviors impact the software quality and how to improve this process for the sake of better software quality. Specifically, this research will focus on modelling the change pattern in code when contributors make the commits and how the code impacts the quality metrics. In this direction, research has been conducted to investigate when, where, how and what the developers contribute to the projects but there is still space for research from the coding side. To reveal details, for example, the change types and contents, in code level can reveal further detail how different changes related to developer behaviors and how they impact the software quality, which is represented in the software quality metrics.

This direction could succeed since the current techniques in machine learning, statistics, natural language processing will be sufficient to support this research. Once it succeeds, we can provide further instructions in coding, if possible, more reliable coding standards which can lead to an improvement and standardization of coding in the software engineering area.

This direction may take more than ten years since it could be a gradual improvement. The success of this direction depends on how the applied techniques evolve in the coming years. The midterm milestone could be a reasonable high prediction accuracy while the final exam milestone could be the completion of the new systematic coding standards.

Research has been done to investigate the impact of some behaviors of developers when they contribute to software, especially open source software. However, in the previous research in this area, they haven't reveal the correlation between the change type and the code as well as their impact on the software quality. In this research, we start with categorization of commit changes in open source software repositories and show their impact on the software quality, collected by using different static analysis tools, by which we get software metrics for each revision of the software. In the end of this paper, we propose a new categorization for the commit change, based on reading the code and the commit message. We also apply neural network to train a model to predict the potential software quality changes after a certain kind of change with an accuracy of 90

In previous research, we have been proven there exist correlation between the change type and the software quality, which is represented by the software quality metrics. However, this is not enough to put this research into valuable application. In this research, we refine the categorization by applying a natural language processing approach to the code to parse it. In this way, we can model the change into a lower level, and make more accurate prediction of what is happening in the repository of the open source software. The results shows we can predict the change of software quality change based on the code with a high accuracy of 95

In previous research, the researchers use natural language processing methods to predict the possible changes to the software quality based on the code changes. This means we can provide guidelines to some extent for software contributors when they commit to a software. However, the previous study mainly focus on training the model with current data which has its limitation. In this research, we collect new data from different software repositories with more than 100 coding styles, including most of existing coding styles. By comparing the different coding styles and training the new prediction model, we come up with a guideline for how to code more efficiently with less software quality issues. The results turn out to be valuable, providing a 95

Acknowledgment

This material is based upon work supported in part by the U.S. Department of Defense through the Systems Engineering Research Center (SERC) under Contract No. HQ0034-13-D-0004 Research Task WRT 1016 – "Reducing Total Ownership Cost (TOC) and Schedule." SERC is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology.

References

[1] P. Behnamghader, P. Meemeng, I. Fostiropoulos, D. Huang, K. Srisopha, and B. Boehm, "A scalable and efficient approach for compiling and analyzing commit history," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '18. New York, NY, USA: ACM, 2018, pp. 27:1–27:10. [Online]. Available: http://doi.acm.org/10.1145/3239235.3239237

[2] F. Hassan, S. Mostafa, E. S. L. Lam, and X. Wang, "Automatic building of java projects in software repositories: A study on feasibility and challenges," in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Nov 2017, pp. 38–47.

[3] F. Hassan and X. Wang, "Change-aware build prediction model for stall avoidance in continuous integration," in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Nov 2017, pp. 157–162.

[4] M. Tufano, F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, A. De Lucia, and D. Poshyvanyk, "There and back again: Can you compile that snapshot?" *Journal of Software: Evolution and Process*, vol. 29, no. 4, p. e1838, 2017, e1838 smr.1838. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.1838

[5] P. Behnamghader, R. Alfayez, K. Srisopha, and B. Boehm, "Towards better understanding of software quality evolution through commit-impact analysis," in *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, July 2017, pp. 251–262.

[6] A. Hindle, D. M. German, and R. Holt, "What do large commits tell us?: A taxonomical study of large commits," in *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, ser. MSR '08. New York, NY, USA: ACM, 2008, pp. 99–108. [Online]. Available: http://doi.acm.org/10.1145/1370750.1370773

[7] A. Alali, H. Kagdi, and J. I. Maletic, "What's a typical commit? a characterization of open source software repositories," in *2008 16th IEEE International Conference on Program Comprehension*, June 2008, pp. 182–191.

[8] N. Dragan, M. L. Collard, M. Hammad, and J. I. Maletic, "Using stereotypes to help characterize commits," in *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, Sep. 2011, pp. 520–523.

[9] E. B. Swanson, "The dimensions of maintenance," in *Proceedings of the 2Nd International Conference on Software Engineering*, ser. ICSE '76. Los Alamitos, CA, USA: IEEE Computer Society Press, 1976, pp. 492–497. [Online]. Available: http://dl.acm.org/citation.cfm?id=800253.807723

[10] A. Mauczka, M. Huber, C. Schanes, W. Schramm, M. Bernhart, and T. Grechenig, "Tracing your maintenance work – a cross-project validation of an automated classification dictionary for commit messages," in *Fundamental Approaches to Software Engineering*, J. de Lara and A. Zisman, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 301–315.

[11] A. Hindle, D. M. German, M. W. Godfrey, and R. C. Holt, "Automatic classication of large changes into maintenance categories," in *2009 IEEE 17th International Conference on Program Comprehension*, May 2009, pp. 30–39.