Quantitative analysis of performance in key parameter of code review - Defects Individuation.

Dorealda Dalipaj[1]

University Rey Juan Carlos, Madrid. dorealda.dalipaj@urjc.es

***Abstract* -** Code review is a well-established and cost-effective way to find defects, employed both in industrial and open source contexts. Today code review is tool assisted, like from Bug tracking systems that were developed to guide maintenance activities of software developers. Thus today it results time efficient and more lightweight compared to the early software inspections performed in the 80s. Finding and removing defects close to their point of injection remains the main motivation and a key parameter for review. However, different studies ([1], [2], [3]) had shown that code review is not performing as expected. They argued that the performance of code review was quite low and that the actual outcome of code review in finding errors was less than expected. Furthermore, another study [4] argues that, as many software programs rely on bug reports to correct software errors in maintenance activities, developers spend much time to identify these bug reports. Thus, the aim of this study is to bring evidence by quantifying the performance of the very key parameter of code review: individuation of bugs. We focus our case study on a cloud computing software, Open Stack [5]. A developer can contribute to its source code repository using the Launchpad Bugtracking [6] and Gerrit Code Review [7]. First, from Launchpad, we individuate the bug reports that are actually describing a defect, and extracted the needed information to bring statistical evidence on the number of defects that are individuated in Open Stack through the years. Second, we compare the first results to the total number of issues reported, grouped by their different types, to have proof of the rate at which the defects are discovered. And last we bring empirical evidence on the effort spent by developers, by means of time, to carry out the defect individuation process.

1 Introduction

Code review, sometimes referred as peer review, is an activity in which people, other than the author of a software deliverable, examine it for defects and improvement opportunities. Code review is characterized *as a systematic approach to examine a product in detail, using a predefined sequence of steps to determine if the product is fit for its intended use* [8]. It is considered one of the most powerful software quality tools available.

There have been different ways of performing defect detenction since its beginning up to nowadays. The formal review or inspection according to Fanagan [9] approach required the conduction of an inspection meeting for actually finding of defects. Different controlled experiments showed that there were no significant differeneces in the total number of defects found when comparing meeting-based with meeting-less-based inspections [10], [11]. Other studies [12] were carried out and proved that more defects were identified with meeting-less-based approaches. As a result a wide range of mechanisms and techniques of code review were developed. From static analysis [15] [16] [17], which examines the code in the absence of input data and without running the code and is tool based, to modern code review [18] [19] [20], which aligning with the distributed nature of many projects is asynchronous and frequently supporting geographically distributed reviewers. Because of their many uses and benefits, code reviews are a standard part of the modern software engineering workflow.

Although it is generally accepted that quality in software remains a challenge due to defects presence. A major quality issue with software is that defects are a byproduct of the complex development process and the ability to develop defect free software remains a big challenge for the software community. It is possible to improve the quality of software product by injecting fewer defects or by identifying and removing defects injected. It is also generally accepted that performance of software review is affected by several factors of the defect detection process. Code review performance is associated with the effort spent to carry out the process and the number of defects found.

A wide set of empirical studies and new approaches have been proposed to understand and improve the review process since it was introduced by Fagan [9]. Sources of process variability range from structure (how steps of the inspection are organized), inspection inputs(reviewer ability and product quality), techniques applied to defect identification that define how each step is carried out), context and tool support [13]. Most empirical studies try to assess the impact of specific process settings on performance. A controlled experiment by Johnson and Tjahjono [10] showed that *total defects identified, effort spent in the process, false positive defects, duplicates* are fundamental variables to analyse when controlling the performance of code review.

2 Discussion

Although code review is used in software engineering primarily for finding defects, severeal studies argue that they often do not find functionality defects.

Microsoft develops software in diverse domains, from high end server enterprise data management solutions such as SQL Server to mobile phone applications and smart phone apps to search engines. Over the past two years, a common tool for code review, Code Flow, has achieved wide-spread adoption at Microsoft. The functionality of CodeFlow is similar to other review tools such Mondrian [18] (adopted at Google), Phabricator [19] (adopted at Facebook) or open-source Gerrit [20]. Two studies where conducted at Microsoft on code review process.

The first study ([2]) took place with professional developers, testers, and managers at Microsoft. The results show that, although the top motivation driving code reviews is finding defects, the practice and the actual outcomes are less about finding errors than expected: Defect related comments comprise a small proportion, only 14%, and mainly cover small logical low-level issues. The second study conducted at Microsoft [3] stated that code review do not find bugs. They found that only about 15% of the comments provided by reviewers indicate a possible defect, much less functionallity issues that should block a code submission.

Another empirical study on the effectiveness of security code review [1], conducted an experiment on 30 developers. They conducted manual code review of a small web application. The web application supplied to the developers had seven known vulnerabilities. Their findings concluded that none of the subjects found all confirmed vulnerabilities (they were able to find only 5 out of 7) and that reports of false vulnerabilities were significantly correlated with reports of valid vulnerabilities.

A different experiment [4] argued that in large scale software programs developers spend much time to identify the bug reports (mainly due to the excessive number of duplicate bug reports).

Keeping in mind the above discussion we did this questions:
1. Are all the code review tools performing a lame number of defects detection? With the abundance of data coming from the engineering systems and having a diverse set of projects to observe [6] [7], we ask if there is any code review providing more value than the others?
2. What is the actual amount of time that developers take in individuating defects?

Work in Progress...

References

1. Edmundson, A., Holtkamp, B., Rivera, E., Finifter, M., Mettler, A., Wagner, D. (2013). An empirical study on the effectiveness of security code review. In Engineering Secure Software and Systems (pp. 197-212). Springer Berlin Heidelberg.

2. Bacchelli Alberto, and Christian Bird. "Expectations, outcomes, and challenges of modern code review." Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, 2013.
3. Czerwonka Jacek, Michaela Greiler and Jack Tilford. "Code Reviews Do Not Find Bugs."
4. Tao Zhang; Byungjeong Lee "A Bug Rule Based Technique with Feedback for Classifying Bug Reports", Computer and Information Technology (CIT), 2011 IEEE 11th International Conference on, On page(s): 336 - 343
5. IaaS cloud computing of Rackspace Cloud and NASA: https://it.wikipedia.org/wiki/OpenStack and https://www.openstack.org
6. Bug Tracking for OpenStack. https://it.wikipedia.org/wiki/Launchpad
7. Gerrit Code Review for OpenStack. //review.openstack.org/Documentation/intro-quick.html
8. D. L. Parnas and M. Lawford. Inspection's role in software quality assurance. In Software, IEEE, vol. 20, 2003.
9. M. E. Fagan. Design and Code inspections to reduce errors in program development. In IBM Systems Journal 15 pp. 182-211, 1976.
10. P. M. Johnson, and D. Tjahjono. Does Every Inspection Really Need a Meeting? In Empirical Software Engineering, vol. 3, no. 1, pp. 9-35, 1998.
11. P. McCarthy, A. Porter, H. Siy et al. An experiment to assess cost-benefits of inspection meetings and their alternatives: a pilot study. In Proceedings of the 3rd International Symposium on Software Metrics: From Measurement to Empirical Results, 1996.
12. A. Porter, H. Siy, C. A. Toman et al. An experiment to assess the cost-benefits of code inspections in large scale software development. In SIGSOFT Softw. Eng. Notes, vol. 20, no. 4, pp. 92-103, 1995.
13. D. E. Perry, A. Porter, M. W. Wade. Reducing inspection interval in large-scale software development. In Software Engineering, I EEE Transactions on, vol. 28, no. 7, pp. 695-705, 2002.
14. Broy M (2002) Ayewah, N., Hovemeyer, D., Morgenthaler, J. D., Penix, J., Pugh, W. (2008). Using static analysis to find bugs. Software, IEEE, 25(5), 22-29.
15. W. R. Bush, J. D. Pincus, D. J. Sielaff. A static analyzer for finding dynamic programming errors, Softw. Pract. Exper. , vol. 30, no. 7, pp. 775802, 2000.
16. Hallem, D. Park, and D. Engler, Uprooting software defects at the source, Queue, vol. 1, no. 8, pp. 6471, 2003.
17. B. Chess and J. West, Secure Programming with Static Analysis, 1st ed. Addison-Wesley Professional, Jul. 2007.
18. N. Kennedy. How google does web-based code reviews with mondrian. http://www.test.org/doe/, Dec. 2006.
19. A. Tsotsis. Meet phabricator, the witty code review tool built inside facebook. http://techcrunch.com/2011/08/07/oh-what-noble-scribe-hath- penned-these-words/, Aug. 2006.
20. Gerrit code review - https://www.gerritcodereview.com/