Poster: Continuous Inspection in the Classroom: Improving Students' Programming Quality with Social Coding Methods*

Yao Lu[1], Xinjun Mao[1], Tao Wang[1], Gang Yin[1], Zude Li[2], and Huaimin Wang[1]
[1]College of Computer
National University of Defense Technology, Changsha, China
{luyao08, xjmao, taowang2005, gangyin, hmwang}@nudt.edu.cn
[2]School of Information Science and Engineering
Central South University, Changsha, China
zli@csu.edu.cn

ABSTRACT

Rich research has shown that both the teaching and learning of high-quality programming are challenging and deficient in most colleges' education systems. Recently, the continuous inspection paradigm has been widely used by developers on social coding sites (e.g., GitHub) as an important method to ensure the internal quality of massive code contributions. In this study, we designed a specific continuous inspection process for students' collaborative projects and conducted a controlled experiment with 48 students from the same course during two school years to evaluate how the process affects their programming quality. Our results show that continuous inspection can significantly reduce the density of code quality issues introduced in the code.

KEYWORDS

Continuous inspection, programming quality, SonarQube

1 INTRODUCTION

College students majoring in Computer Science and Software Engineering need to master skills for high-quality programming [3], that is, the ability to write code with high readability, understandability, maintainability, etc. However, previous studies have shown that both the teaching and learning of high-quality programming skills are challenging and deficient in most colleges' education systems [3]. It is clear that the lack of high-quality programming capabilities puts job-hunting students at a disadvantage and can even influence the productivity and quality of newly hired employees.

Social coding sites, such as GitHub, receive large numbers of contributions from developers around the world, and the frequent turnover of external developers and the wide variations among their coding experience challenge the project management of code and its internal quality [1]. To meet this challenge, the continuous inspection method is increasingly adopted and integrated into the contribution workflow [2]. The method implements continuous and automated inspection processes on the internal quality of incoming pull requests (PRs). In the process, a PR will be automatically rejected if it does not meet static quality standards. From the perspective of contributors, the results of continuous inspection identify their bad coding habits. In addition, the continuous alerts and modifications in the process can help them to learn and memorize their quality issue patterns according to the *Ebbinghaus Forgetting Curve*. This motivates us to examine whether such an automated quality assurance method can be used in the classroom setting to address the challenges in the teaching and learning of high-quality programming.

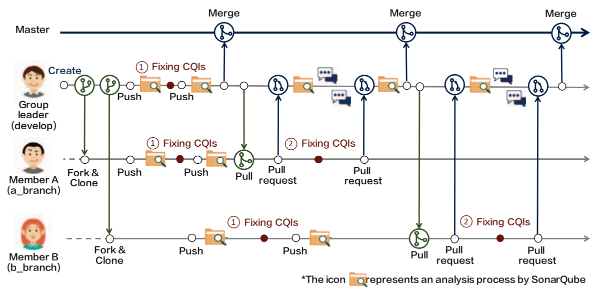

Figure 1: The continuous inspection process for students' collaborative projects

2 THE CONTINUOUS INSPECTION PROCESS

Based on the continuous inspection process model in GitHub, we design a quality assurance process by integrating continuous inspection into students' collaborative projects. The process is based on the git branching collaboration model. The process (see Figure 1) begins with the team leader creating the project repository. Then, each member forks the repository and clones it to her local git repository. The team leader submits code on the *develop branch*, and each member develops on her own branch. After making changes, *e.g.*, implementing a feature or fixing a defect, a member pushes her local changes to the remote repository. The quality analysis process is then automatically triggered, and the member should fix the code quality issues (CQIs) reported by the static quality analysis tool (see mark ① in Figure 1). When she decides to merge her contributions to the leader's branch, she first pulls the latest code on the *develop* branch to resolve the conflicts caused by concurrent

*This research is supported by research grants from Natural Science Foundation of China under Grant No. 61472430, 61502512, 61532004 and 61379051.

ICSE'18, May 2018, Gothenburg, Sweden
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.475/123_4

changes. Then, she opens a PR, requesting the leader to merge the branch containing her changes into the *develop* branch. The team leader and members can discuss and review the PR. When the PR is accepted and merged to the *develop* branch, the quality analysis process is triggered, which can again lead to the CQI fixing actions (see mark ② in Figure 1). In addition, we stipulate the following regulations that the students should follow during the process: (1) each team should fix all CQIs of the initial analysis process in the first two days of the semester; (2) each student is responsible for fixing the CQIs of her modules on her own branch; (3) the CQIs should be fixed within 24 hours after being reported by SonarQube; (4) all CQIs on the *master* branch should be fixed or closed. If a CQI is marked as *'won't fix'*, reasons should be given as comments in SonarQube.

3 THE EXPERIMENT

We conducted the experiment on the students in the same courses in the 2016 and 2017 school year respectively. We had 22 (grouped into 5 teams) and 26 students (6 teams) in the 2016 school year and 2017 school year, respectively. In our college, the undergraduates majoring in Software Engineering have to take two mandatory practice courses, which are spread over the spring and summer semesters respectively. In the courses, the students have to collaboratively develop a well-designed ingenious software project using the *Iterative Development Model*. They are required to accomplish four iterations (two iterations in each semester). After the first two iterations of the course project in the spring semester, the students are usually more familiar with the programming language, development environment and pull-based collaborative development process. Therefore, in the summer semester, the students are required to execute the continuous inspection process through the final two iterations. We leverage *TRUSTIE*[1] [4] to execute the continuous inspection process. We use the automation server Jenkins and the quality analysis tool SonarQube to implement an automated process to analyze the latest revision in the code repositories when new code changes are pushed or merged to the GitLab server. After installing the Git plugin, SonarQube can automatically detect the introduced commit of a CQI using the *git blame* command. This feature allows us to easily obtain the CQIs introduced by a student.

We designed the controlled experiment from two dimensions. The first dimension examined the programming-quality changes of the student individuals before and after they participated in the continuous inspection process. Second, to examine whether the effects (if any) were caused by the process, we compared student CQID changes between two groups that did adopt and did not adopt continuous inspection. Because the number of the teams in 2016 was odd, they all participated in the continuous inspection process. Accordingly, in 2017, we randomly divided the 6 teams into two groups (3 teams in each group): the control group, which adopted continuous inspection, and the experimental group, which did not.

4 RESULTS

We used the introduced *CQID*, that is, the number of introduced CQIs per changed code line, to measure programming quality [1]. Given that all the projects used Java programming language, we

[1]https://www.trustie.net/login

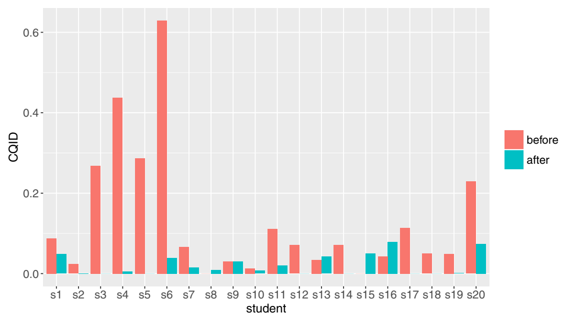

Figure 2: Students' introduced CQIDs before and after adopting continuous inspection

only analyzed the code quality of the Java modules. We first compared the introduced CQIDs of the students who adopted continuous inspection between the two phases, i.e., the spring semester when they did not adopt continuous inspection and the last quarter of the summer semester when they did adopt continuous inspection. Ten students in the 2016 and 2017 school years submitted Java code in both phases. The introduced CQIDs in the two periods are shown in Figure 2: *s1-s10* represent the students in 2016 and the remaining represent students in 2017. The results show that the number of introduced CQIDs significantly decreased for the majority of the students after adopting the continuous inspection process. Moreover, the results of the *Wilcoxon signed-rank paired test* for the students in both school years confirmed the significance: p values of 0.01 and 0.049 for the 2016 and 2017 school years, respectively (p<0.05). Furthermore, we conducted the same analysis on the experimental group who did not adopt continuous inspection in the 2017 school year, and the results did not show significant differences between introduced CQIDs (p value of 0.371). Therefore, we conclude that the number of introduced CQIDs significantly decreased after adopting the continuous inspection process.

5 CONCLUSION

In this study, we introduced continuous inspection, an internal quality assurance method widely adopted on social coding sites, into an educational setting. We designed a specific continuous inspection process for students' collaborative projects and then conducted a two-year controlled experiment to investigate how the process influences the students' programming quality. On the basis of the data, we found that the process can significantly reduce the CQIDs introduced in the code.

REFERENCES

[1] Yao Lu, Xinjun Mao, Zude Li, Yang Zhang, Tao Wang, and Gang YIn. 2016. Does the Role Matter? An Investigation of the Code Quality of Casual Contributors in GitHub. In *The 23rd Asia-Pacific Software Engi. Conf. (APSEC 2016)*. 49–56.
[2] Yao Lu, Xinjun Mao, Zude Li, Yang Zhang, Tao Wang, and Gang Yin. 2017. Internal quality assurance for external contributions in GitHub: An empirical investigation. *Journal of Software Evolution & Process* 50 (2017), e1918.
[3] Mamoun Nawahdah and Dima Taji. 2016. Investigating students' behavior and code quality when applying pair-programming as a teaching technique in a Middle Eastern society. In *IEEE Global Eng. Educ. Conf.*
[4] Huaimin Wang, Gang Yin, Xiang Li, and Xiao Li. 2015. *TRUSTIE: A Software Development Platform for Crowdsourcing*. Crowdsourcing. Springer Berlin Heidelberg. 165–190 pages.