Measuring the impact of code quality analysis tools
Work in Progress

Julian Jansen[1,2,3], Ana Oprescu[2], and Magiel Bruntink[3]

[1]julian.jansen@gmail.com
[2]University of Amsterdam
[3]Software Improvement Group

Abstract

While some university-level programming courses focus on software quality, often in introductory courses code quality is little touched upon due to time constraints. When students work on a programming assignment, they usually get feedback on the code quality after the assignment was graded. This feedback cannot be used on that same assignment, before the grade is determined. Better Code Hub is a service that checks code quality according to ten guidelines. We employ Better Code Hub as a formative assessment and feedback tool that students can use to monitor their progress on code quality. Our aim is to improve students' skills for code quality during the evolution of a students' programming assignment, while keeping the overhead low for teaching staff as well as for students. Preliminary findings indicate that there is an improvement in the code quality of the students' assignments over the period Better Code Hub was used.

This research is performed at the Software Improvement Group as a thesis project for the Software Engineering master program of the University of Amsterdam.

Proceedings of the Seminar Series on Advanced Techniques and Tools for Software Evolution SATToSE 2017 (sattose.org). 07-09 June 2017, Madrid, Spain.

1 Introduction

> "A strong digital economy is vital for innovation, growth, jobs and European competitiveness. The spread of digital is having a massive impact on the labour market and the type of skills needed in the economy and society.[...] It is leading to the need for more skilled ICT professionals in all sectors of the economy. It is estimated that there will be 500,000 unfilled vacancies for ICT professionals by 2020." ([2])

Education in programming is a means to fill this gap between the supply and demand of skilled programmers. At the University of Amsterdam (UvA), in a programming minor with courses not belonging to a bachelor program[1], a student starts without any programming knowledge. During a semester, the student starts with learning the fundamentals of programming and later on specializes in making mobile applications or data visualisations. Part of the skills of an ICT professional is the ability to write code that is of high quality. During the minor, there is little to none systematic teaching in the rationale behind code quality. Most of the educational effort is spent on the aspects of learning the fundamentals of programming, new languages, and development environments. Given the constraints in time and the focus on these three aspects, students get elaborate feedback on their code quality only at the end of the assignment. This feedback is given after the grading and cannot be used during the assignment itself. Also, the grade depends on code quality criteria.

This research focuses on measuring the impact of a tool that helps novice programmers improve their code quality. Better Code Hub (BCH) is such a tool,

[1]`http://www.mprog.nl/`

developed by the Software Improvement Group (SIG) based on their maintainability model [4]. BCH provides the user with feedback based on the ten guidelines shown in Table 1. It presents refactoring candidates and a short text of how and why the refactorings should be done. The ten guidelines can be divided into three categories, as shown in Table 1. The BCH website (`www.bettercodehub.com`) provides an explanation per guideline.

Through the following research questions, we emphasize that there is a responsibility of transferring the knowledge of why refactorings should be done (the rationale). Also, it is not only about impacting how the grading is done, rather the entire process.

Research question 1: What kind of impact does an automatic assessment tool have on the students' skills to write better quality code over the span of a learning unit?

Research question 1.1: How can we measure the impact of an automatic assessment tool on the code quality of students' assignments?

Research question 2: To what extent do BCH and the rubric overlap with respect to the rubric's criteria?

We will discuss teaching code quality, and tools to provide feedback to students. This is followed by our aim, research method, and the first results of our experiment. Preliminary findings indicate that there is an improvement in the code quality of the students' assignments over the period BCH was used.

2 Background

Our research focuses on teaching students about code quality as a part of the larger topic of software maintenance. Currently, grading is outside the scope of our work. Teaching practical software maintenance can be done in different ways. The *Software Evolution* course of the *Software Engineering* master of the UvA has groups of students analyse an open source code base with tools they developed themselves based on the SIG maintainability model [4]. A recent study proposes to enhance group-based maintenance assignments, that usually focus on code bases that are small and of very good quality, in which artificial problems are introduced [8]. They propose to use old medium-sized student projects with real software bugs. However, these are courses where maintenance is the main topic of the curriculum, and the students are somewhat experienced programmers.

Stegeman *et al.* [7] discuss the design of a feedback rubric for students of introductory programming courses. Their rubric is based on criteria from their code quality model [6], and describes *what* is expected from these criteria. The rubric can be augmented to be used as a formative assessment tool, to provide feedback to students to improve their performance. For example, students can review each other's code or use it for self-reflection.

An overview of automated assessment approaches is provided in [1]. Many of the presented tools in [1, p. 99] are developed for local use, where BCH can be employed as long as the programming language is supported. As stated in [1, p. 95], the type of feedback naturally affects the working strategy of the students. Formative feedback can be defined as "information communicated to the learner that is intended to modify his or her thinking or behavior to improve learning [5, p. 154]." This is feedback that can be used during an assignment, or used in the next assignment within a course. We understand summative feedback as feedback that is provided after the last assignment of a course. This means that students cannot employ feedback during the programming assignments of a course, before the final assessment. As stated by [5, p. 154], the crux is how feedback can be delivered correctly to significantly improve the learning process and outcomes.

3 Aim

The main goal of education is transferring knowledge and skills. So how do we transfer knowledge about code quality to beginning programmers? Firstly, it should be defined what code quality means within the domain of programming education. What corpus of theory is relevant within this domain? This will be answered by doing a literature survey and review comparable university-level courses. Our next goal is to improve students' skills for code quality during the evolution of a student's programming assignment, while keeping the overhead low for both teaching staff and students. We hypothesize that the more analysis runs are performed with an automatic recommender-assessment tool, such as BCH, the more code quality guidelines are met.

The overall contribution of this work, is to give insight into the impact of introducing code quality tools into an educational setting. We are not just validating a tool. We are trying to find ways in which automatic assessment tools can help the educational process: provide students with feedback on how to improve their code quality **while** doing their assignments, not **after** the assignment was finished. Also, by introducing this tool into the context of a pre-existing course, we are looking for problems we can solve with the tool.

Table 1: 10 guidelines for code quality divided into categories.

Code	Architecture	Way of Working
Write Short Units of Code	Separate Concerns in Modules	Automate Tests
Write Simple Units of Code	Couple Architecture Components Loosely	Write Clean Code
Write Code Once	Keep Architecture Components Balanced	
Keep Unit Interfaces Small	Keep Your Codebase Small	

4 Research method

We are doing a case study, where we have access to students in a situation where we cannot control all variables. As stated in [3, p. 301], major changes in educational strategies can only be studied when implementing them, leading us to *action research.*

In this work, we use the Technical Action Research (TAR) method [10, 11]. TAR is an approach to validate new artefacts under conditions of practice. TAR starts with an artefact that is tested under conditions of practice by solving concrete problems with it [10, p. 220]. In this case, the artefact is BCH and the practice is in an educational setting. TAR bridges the relevance gap between "idealizations made when designing the artefacts and the concrete conditions of practice that occur in real-world problems [10, p. 220]."

TAR starts with the design of an artefact, and then looks for organizational problems that could be solved with the artefact. This makes TAR an artefact-driven approach, where other forms of *action research* are problem-driven. In our case SIG is developing the artefact, and we as researchers are validating BCH by looking for problems we can solve with this tool.

The goal of the researcher is "to develop this artifact for use in a class of situations imagined by the researcher [10, p. 221]." The artefact is first tested on toy-problems under idealized conditions. Afterwards, it is scaled up to conditions of practice and more realistic problems are solved with it. This is done until it can be tested by using it in one or more concrete client organizations to solve concrete problems [10, p. 221]. In action research an intervention in a social situation is done in order to both improve this situation and learn from it [10, p. 220].

To validate our treatment of using BCH as a formative feedback tool, we perform at least five TAR cycles. Two will be performed within the context of the course "Programmeerproject", two in the context of "App Studio" and one in the context of "Programmeertheorie". [2]

5 Evaluation

The first TAR cycle was performed in January of 2017. "Programmeerproject" is a four-week full-time programming course where the student creates a mobile application (*Android* or *iOS*) or a web visualization (*D3.js*) of their own choice (adhering to some requirements). The first week opened with an introduction-lecture, and a guest-lecture about maintainability and BCH. In the first week a prototype is created, followed by an alpha version in the second week. In the third week, the goal is to develop a fully functional beta version, which needs to be put through BCH at the end of the week (if not already done). In the fourth week, the goal is to refactor the code and do the last bug fixes. The students are free to use BCH and have the accompanying book [9] available to them the classroom. BCH is not used for grading, and the teaching staff is oblivious to how many times each student uses BCH.

To answer research question 1 and 1.1, we need data on how running BCH analysis impacts the code produced by students. And to answer research question 2, we need to compare against traditional feedback, obtained from a source that is BCH-agnostic.

At this stage of the research we depict the data of the first TAR cycle. We monitored a total of 66 students in the course and 52 of them used BCH. They were divided over three tracks: 16 in the *Android* track, 21 in the *Data* track and 15 in the *iOS* track.

5.1 Preliminary results

Each time the student analyses her assignment, a data point is created containing information about which guidelines are met. The improvement on the ten guidelines (as listed in Table 1) is plotted in Figure 1a. The improvement on the four code guidelines is plotted in Figure 1b.

On the x-axis, the first and last use of BCH of every project is matched up. For example, a project that analyses the code only two times over a period of 1 day is mapped in the range [0.00 . . . 1.00]. A project that is analysed 100 times over the course of 2 weeks is similarly mapped. In this way the BCH usage period is aligned. The 0.00 marks the starting score and the 1.00 marks the end score in BCH. The y-axis shows the amount of guidelines that are met, with zero indicating that no compliance with the guidelines is met.

To correct for the measurement frequency of a user, which can affect the trend line, the data points are

[2]For course information see `http://www.mprog.nl/vakken/`

interpolated by taking an approximation at each relative time unit. Only students that used BCH more than two times are included. This filtering is based on the reasoning that running BCH less than three times is not considered active use. Using BCH will not provide a measure of improvement. Running BCH only two times will only provide a data point at the beginning and in the end, and will result in only interpolated data points in between.

Figure 2 plots the number of runs of BCH per student against the end score on the four code guidelines. The y-axis indicates the amount of guidelines that are met, with zero indicating that no compliance with the guidelines is met. Figure 2a shows an outlier of 117 runs. In Figure 2b we filter out this data point to get a closer look at the first half of the graph.

6 Discussion

Figure 1a shows that there is an improvement from the start of using BCH until the last run. The improvement on the guidelines that fall under the code category in Table 1, is depicted in Figure 1b. The data for the ten guidelines and the four guidelines show a positive trend in code quality over time.

We hypothesize that the more analysis runs are performed with BCH, the more guidelines are met. Figure 2 suggests that the biggest gain in quality is achieved within the first ten runs. We do not want to draw any hard conclusions yet from this graph, and wait until more data from the remaining TAR cycles are added, because newly added data points after the ten runs mark might change the depiction.

7 Limitations

Since for ethical reasons we cannot have two groups within the same course, out of which only one using BCH, we set out to compare the data of this TAR cycle against the outcome of the same course in the previous iteration without BCH. Our aim is to show that the improvements in code quality can be attributed to a formative assessment tool.

Researcher bias might be introduced by presenting the data in a favourable way. A way to reduce this is via critical reflection [3, p. 302]. Also, the interpretations will be validated by performing multiple TAR cycles.

SIG developed BCH and might be able to use it in a way no one else can. This would imply a lack of generalizability [10, p. 232]. This will be mitigated by teaching others to use BCH in the form of guest-lectures and manuals aimed at teaching assistants and students.

Other data gathering strategies like surveys will be used. A threat might be that the stakeholders answer the interview questions in a socially desirable way or that we interpret the answers in a desirable way [10, p. 232-233]. We can mitigate this threat by having a third-party do the data collection and coding. This is most applicable to the interpretation of the rubrics.

8 Conclusions and future work

We cannot yet draw hard conclusions from the amount of data we have at this point, however the preliminary results look promising. Verbal feedback from the students and teaching staff in the first TAR cycle gave us confidence to ramp up the data gathering efforts, by deploying BCH in more courses.

The second TAR cycle is completed and the data is currently being gathered. We need to compare this new data to the data of the first TAR cycle. Additionally, we need to find out what data can be joined. Also, the knowledge gained from the first two cycles must be formalized and used to improve the two TAR cycles that just started.

Besides gathering data via BCH and the rubrics, we can gather more data via surveys. For example, the evaluation form that is filled out at the end of a course. Proper questions need to be constructed.

At the time of writing, the next step is to correlate the data from BCH with the data from the rubrics. Can we find correlations between the guidelines of BCH and the code quality criteria from the rubric?

Acknowledgements

We would like to thank Martijn Stegeman (UvA), Gracia Redder (UvA), Marco Di Biase (SIG) and Paco van Beckhoven (UvA / SIG) for their constructive feedback on this submission.

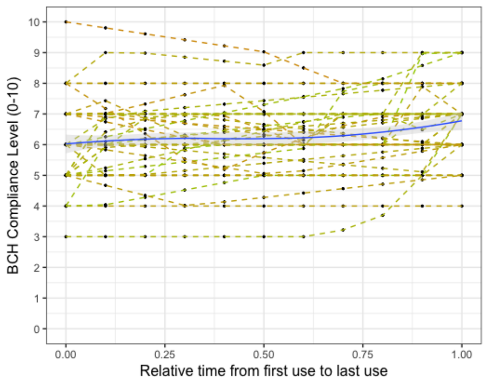
References

[1] Kirsti M Ala-Mutka. A survey of automated assessment approaches for programming assignments. *Computer science education*, 15(2):83–102, 2005.

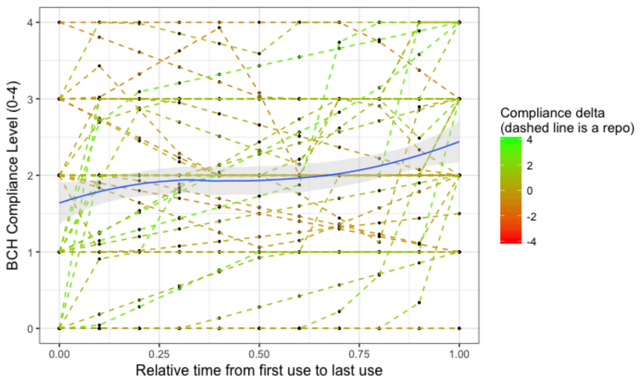
[2] European Commission. Digital skills & jobs. Website, June 2016. `https://ec.europa.eu/digital-single-market/en/skills-jobs`, accessed April 2017.

[3] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering*, pages 285–311. Springer, 2008.

[4] Ilja Heitlager, Tobias Kuipers, and Joost Visser. A practical model for measuring maintainability. In *Quality of Information and Communications Technology, 2007. QUATIC 2007. 6th International Conference on the*, pages 30–39. IEEE, 2007.

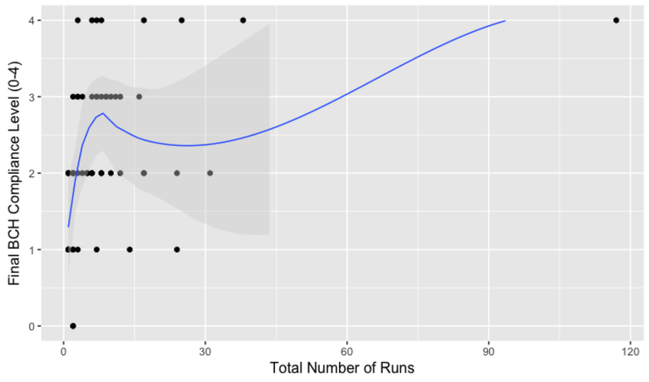


(a) Improvement on all ten guidelines.

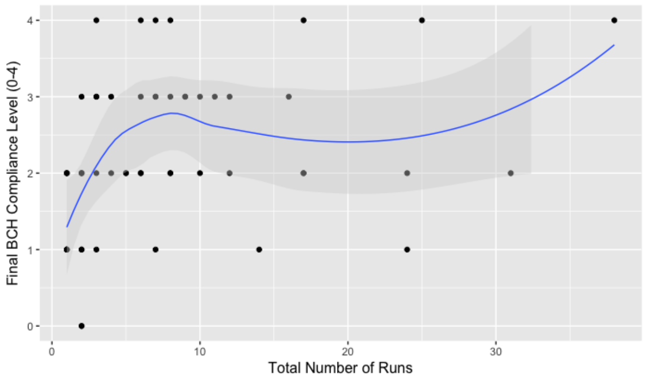


(b) Improvement on the four code guidelines.

Figure 1: First and last analysis are synced up. Zero indicates that no compliance is met with the guidelines. The data points are interpolated by taking an approximation at each relative time unit. Only includes students with more than two runs.



(a) Scores of all runs.



(b) Scores under 60 runs.

Figure 2: Final compliance score plotted against the total runs of BCH. Zero indicates that no compliance is met with the guidelines.

[5] Valerie J Shute. Focus on formative feedback. *Review of educational research*, 78(1):153–189, 2008.

[6] Martijn Stegeman, Erik Barendsen, and Sjaak Smetsers. Towards an empirically validated model for assessment of code quality. In *Proceedings of the 14th Koli Calling International Conference on Computing Education Research*, pages 99–108. ACM, 2014.

[7] Martijn Stegeman, Erik Barendsen, and Sjaak Smetsers. Designing a rubric for feedback on code quality in programming courses. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*, pages 160–164. ACM, 2016.

[8] Claudia Szabo. Student projects are not throwaways: teaching practical software maintenance in a software engineering course. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 55–60. ACM, 2014.

[9] Joost Visser, Sylvan Rigal, Rob van der Leek, Pascal van Eck, and Gijs Wijnholds. *Building Maintainable Software, Java Edition: Ten Guidelines for Future-Proof Code.* " O'Reilly Media, Inc.", 2016.

[10] Roel Wieringa and Ayşe Moralı. Technical action research as a validation method in information systems design science. In *International Conference on Design Science Research in Information Systems*, pages 220–238. Springer, 2012.

[11] Roel J Wieringa. Technical action research. In *Design Science Methodology for Information Systems and Software Engineering*, pages 269–293. Springer, 2014.