Integrating Code Reviews into Online Lessons to Support Software Engineering Education

Juan Carlos Farah[1], Basile Spaenlehauer[1], María Jesús Rodríguez-Triana[2], Sandy Ingram[3], and Denis Gillet[1]

[1] École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland,
{juancarlos.farah,basile.spaenlehauer,denis.gillet}@epfl.ch
[2] Tallinn University, Tallinn, Estonia,
mjrt@tlu.ee
[3] University of Applied Sciences (HES-SO), Fribourg, Switzerland,
sandy.ingram@hefr.ch

Abstract. The use of peer code review exercises is well established in software engineering education. Nevertheless, challenges involving students' ability to perform code reviews have been identified as barriers to successfully integrating code reviews in educational settings. We have previously proposed code review notebooks as a way to address this issue. Code review notebooks resemble computational notebooks but focus on reviewing rather than executing code and can serve to introduce students to the code review process. In this study, we evaluated the effects of using a code review notebook via a case study whereby 25 university students were taught how to identify code style issues in JavaScript. Our mixed-method analysis suggests that the code review notebook format encourages students to reflect on their learning process and can result in short-term learning gains. These findings could serve instructors looking to incorporate code review exercises into their practice.

Keywords: code review, software engineering education, online lesson

1 Introduction

Code reviews—the process whereby code written by one programmer is cross-checked by another programmer for potential bugs or issues—are standard practice in software engineering. Most studies addressing the use of code reviews in education focus on peer code reviews, in which students complete a programming assignment and review each other's submissions [12]. Peer review has been championed as a way to teach students to provide and receive constructive criticism [2], as well as to build collaboration skills [24]. Furthermore, the rising popularity of social coding platforms (e.g., GitHub, GitLab) as pedagogical tools has resulted in students being exposed to professional code review tools and processes as part of their education [10]. Nevertheless, one of the challenges of using peer code review in education is the fact that students often lack the ability to perform code reviews [12]. In this paper, we present the evaluation of an online lesson format aimed at teaching students how to perform code reviews. We refer to lessons following this format as *code review notebooks*, as they resemble computational notebooks, but focus on performing code reviews and not on executing code. By completing lessons featuring code review notebooks, students could both learn about the code review process through example code snippets and get practical experience through interactive exercises.

In a previous study [8], we assessed the usability of the code review notebook format, achieving positive results. In this study, we focus on the format's effectiveness in achieving short-term learning gains, encouraging students to reflect on their learning process, and providing a satisfactory learning experience. We conducted a case study comprising a within-subjects experiment with 25 university students. The findings from our mixed-method analysis suggest that code review notebooks could serve as a way to introduce students to software engineering concepts such as programming best practices, as well as to illustrate how the code review process works. Students would therefore be better equipped to participate in peer code review exercises. Through this evaluation, we aim to consolidate the design of online lessons aimed at preparing students for the code review process as a part of their software engineering education.

2 Background and Related Work

In a recent systematic literature review, Indriasari *et al.* reported that the first "use of peer code review in a programming course was published in 2003", with 187 studies published as of April 2019 [12]. Indriasari *et al.* also emphasize that the most common approach to incorporating code reviews into educational contexts is to include them as a peer review exercise. To provide the technological context for these peer code review exercises, instructors often integrate social coding platforms or bespoke tools (e.g., CaptainTeach [18], EduPCR [26], Caesar [23]), specifically designed for peer code review. These peer code review tools provide functionalities that are particularly useful to instructors, such as code partitioning and automatic review assignment [23]. Indeed, Indriasari *et al.* underline that "[one] of the main benefits of peer code review, particularly in large classes, is to assist the instructor in providing timely feedback to students" [12]. Nevertheless, properly motivating and training students to provide feedback is a challenge that these same authors identify as a barrier to the successful adoption of peer code review exercises in the classroom. If students are expected to review code submitted by their peers, they should be able to do so satisfactorily, as otherwise the exercise is neither useful for the student conducting the review nor for the student receiving the feedback. To address this challenge, a few studies have proposed focusing on teaching the code review process to individual students, namely, work by Ardıç *et al.* [3], Ribeiro Guimarães [19], and Song *et al.* [21]. Our work is most aligned with this line of research.

In a previous study [8], we presented a code review application that could be used to scaffold code review notebooks. Our evaluation using standard instruments for measuring user experience (System Usability Scale [5] and the short version of the User Experience Questionnaire [20]) suggested that students rated the usability of the code review notebook and the underlying code review application positively. The current study builds on these results, this time addressing the potential code review notebooks have to support learning experiences for software engineering students.

3 Methodology

The purpose of our evaluation was to address one main research question:

> *What are the effects of using a code review notebook on students' experiences learning to identify code quality issues in JavaScript?*

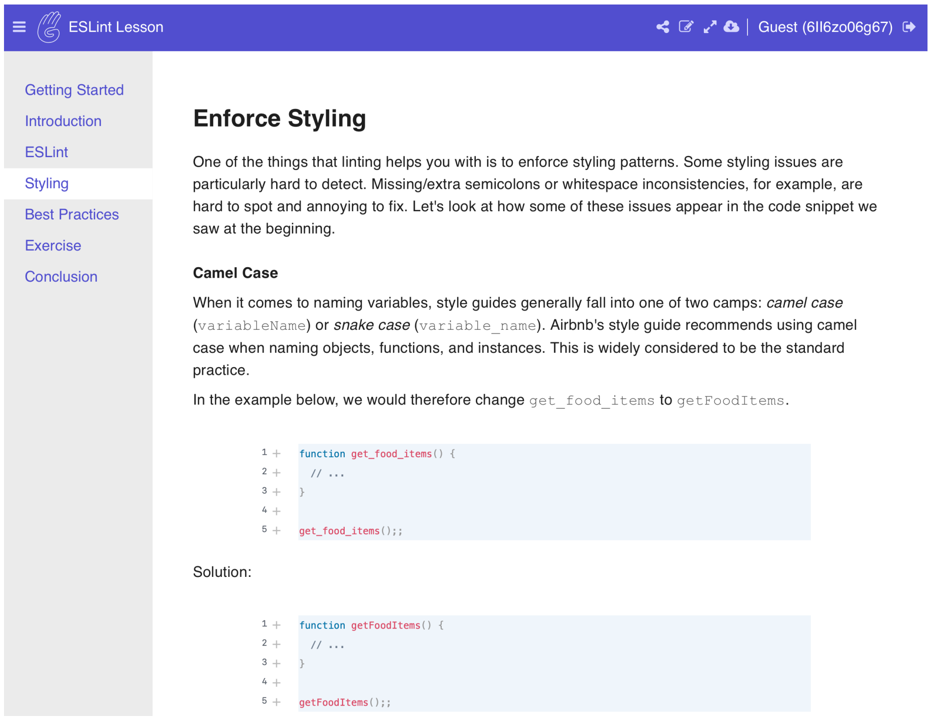
To frame our evaluation, we focused on three aspects of the learning experience: (i) learning gains achieved, (ii) self-reflection on the learning experience, and (iii) feedback regarding the lesson. We evaluated our code review notebook through a case study focusing on these three aspects. Our study consisted of a within-subjects experiment that took place in an online learning environment. In this section, we present the methodology used for our evaluation.

3.1 Context

To perform our evaluation within an educational setting, the context for our case study was an asynchronous online lesson[4] on code linting. The process of linting code dates back to the 1970s and involves the use of static analysis tools designed to detect issues in software [13]. Linting tools are often used in industry to enforce code quality standards and help automate code reviews. It is therefore pertinent to teach the concept of code linting using a code review notebook, especially since the code review process has also been proposed as a pedagogical tool for introducing students to code quality standards [15]. In our case, we specifically introduced students to ESLint [25], a linting tool designed for JavaScript, as well as the Airbnb JavaScript Style Guide [1], a popular configuration for ESLint. In the following sections, we highlight the technological and pedagogical contexts used to carry out our study in an educational setting.

Technological Context Our code review notebook was implemented on the Graasp digital education platform [9]. Graasp allows instructors to create online learning experiences featuring a varied set of web-compatible resources. Instructors can therefore prepare an online lesson comprising textual explanations alongside multimedia content and interactive applications. To prevent overwhelming students with content, instructors can divide the lesson into phases, which the students can explore using a vertical navigation bar. For our implementation, we followed the code review notebook format. Textual explanations

[4] The online lesson is available here: https://graasp.eu/s/sm12t9.



Enforce Styling

One of the things that linting helps you with is to enforce styling patterns. Some styling issues are particularly hard to detect. Missing/extra semicolons or whitespace inconsistencies, for example, are hard to spot and annoying to fix. Let's look at how some of these issues appear in the code snippet we saw at the beginning.

Camel Case

When it comes to naming variables, style guides generally fall into one of two camps: *camel case* (`variableName`) or *snake case* (`variable_name`). Airbnb's style guide recommends using camel case when naming objects, functions, and instances. This is widely considered to be the standard practice.

In the example below, we would therefore change `get_food_items` to `getFoodItems`.

```
function get_food_items() {
  // ...
}

get_food_items();;
```

Solution:

```
function getFoodItems() {
  // ...
}

getFoodItems();;
```

Fig. 1. Code quality issues were explained using text accompanied by a code snippet containing the issue and another snippet illustrating a possible solution.

regarding the code review process and code linting were intercalated with images and code snippets. Code snippets were presented using the code review application [8], which allowed students to annotate each line of code with comments. Furthermore, a text input application was used to allow students to answer open-ended questions. This online lesson—depicted in Fig. 1—was shared with students via a link, which they could access anonymously.

Pedagogical Context Our online lesson was structured following the *Fixer Upper* pedagogical pattern, whereby students are presented with code "that is generally sound but [contains] carefully introduced flaws [that] can both introduce a complex topic early and serve as a way to introduce error analysis and correction" [4]. The lesson was structured into seven phases that students were meant to navigate in order. The first phase (*Getting Started*) introduced the first code snippet that would serve to highlight the code style issues that would be addressed in the lesson. Students were presented with an exercise comprising a code snippet and were asked to annotate it with any potential issues they could identify. This initial exercise served as a pre-test to gauge the student's initial knowledge of code reviews, linting, and JavaScript best practices. In the second phase (*Introduction*), we formally introduced the lesson with a short textual introduction to linting. The third phase (*ESLint*) presented ESLint and the Airbnb JavaScript Style Guide. The fourth (*Styling*) and fifth (*Best Practices*) phases made up the core of the lesson. In these phases, we used the code quality issues seeded into the pre-test as examples and walked the student through them using (i) an explanation based on the ESLint documentation and Airbnb JavaScript Style Guide, (ii) a code snippet highlighting the issue, and (iii) a code snippet illustrating a possible solution. Fig. 1 depicts the fourth phase, highlighting an explanation block featuring the code review application. In a sixth phase (*Exercise*), we presented an exercise that served as a post-test to gauge students' short-term learning gains. A seventh phase (*Conclusion*) served as a wrap-up, presenting the answers to the post-test and asking students to provide (i) a short reflection on their performance and (ii) feedback regarding the lesson.

3.2 Participants

We recruited 27 students pursuing degrees in technical subjects via our internal networks at the École Polytechnique Fédérale de Lausanne and the School of Engineering and Architecture of Fribourg in Switzerland. Students were informed that this was an optional, ungraded exercise. Two students did not complete the post-test and were therefore excluded from our analysis. A total of 22 students—9 female and 13 male—completed an optional demographics questionnaire. Of these students, 17 were currently completing a master's degree, while 5 were enrolled in a bachelor's program. Students were also asked to report on a scale of 1 to 5—with 1 being *Beginner* and 5 being *Expert*—their programming experience both overall and specifically using JavaScript. Results show that while students' programming experience was above average ($\bar{x} = 3.18$, Mode = 4), they were mostly uninitiated in JavaScript ($\bar{x} = 1.95$, Mode = 1).

3.3 Procedure

Our study took place in January 2022 and was conducted remotely. Students were sent a link to the code review notebook described in Section 3.1 and were instructed to take a total of 30 minutes to complete the evaluation. Students accessed the online lesson anonymously and completed an initial exercise that served as a pre-test before going through the code review notebook and then taking a post-test. Upon completion of the online lesson, students were directed to an online questionnaire that included an optional demographics section.

3.4 Instruments

To address our research question, we operationalized the first aspect—learning gains—using a bespoke instrument based on data from the pre- and post-tests. Learning gains were calculated as the difference between a student's post- and pre-test scores, with a possible range from -100% to 100%, inclusive. The second aspect—self-reflection—was addressed via a question[5] asking students to provide feedback on their performance in short answer form. Finally, the third aspect—feedback—comprised an open-ended question[6] asking students to provide their thoughts on the lessons as well as any comments or suggestions.

3.5 Data Analysis

The dataset used for our analysis was extracted from the Graasp platform using its learning analytics pipeline [7]. To analyze our data, we followed a mixed-method approach. Quantitative data were analyzed using descriptive and inferential statistics. Specifically, we report the sample mean ($\bar{x}$), median ($\tilde{x}$), and standard deviation (s), and use a dependent t-test for paired samples to compare the distributions of pre-test and post-test scores. Qualitative data were analyzed using line-by-line data coding [6]. We also performed sentiment analysis on the self-reflection and feedback comments. For this analysis, we used VADER [11]—a sentiment analysis model trained on social media data—to assign a sentiment score ranging from -1 to $+1$ to each comment.

4 Results

In this section, we outline our results following the three aspects of our study.

4.1 Learning Gains

In the pre-test, students detected 14.5% of issues on average ($\bar{x} = 14.5\%, \tilde{x} = 9.09\%, s = 13.1\%$), while in the post-test, the mean increased to $\bar{x} = 48.6\%$ ($\tilde{x} = 50.0\%, s = 23.1\%$). The differences between the pre- and post-test resulted in a mean learning gain of $\bar{x} = 34.0\%$ ($\tilde{x} = 35.7\%, s = 20.0\%, x_{\text{min}} = -1.95\%, x_{\text{max}} = 71.4\%$), with all students except one achieving positive learning gains. To assess the distribution of learning gains (and therefore the distribution of the differences between the pre- and post-test) we performed a Shapiro-Wilk test, which did not show evidence of non-normality ($W = 0.967, p = 0.578$). Indeed, Fig. 3 depicts a histogram of the learning gains, visually suggesting a normal distribution. A dependent t-test for paired samples showed that there is a significant difference in the distribution of the pre- and post-test scores ($t(24) = 8.51$, $p < 0.001$). Fig. 2 illustrates the differences between both distributions.

4.2 Self-Reflection

A total of 23 students provided a short open-ended answer to a question asking them to reflect on their performance in the post-test. Comments ranged from

[5] "Did you manage to find all of these [issues]? If not, which ones did you miss? Did you find any of them particularly tricky/helpful?"

[6] "What did you think about this lesson? Any comments, suggestions, or feedback?"

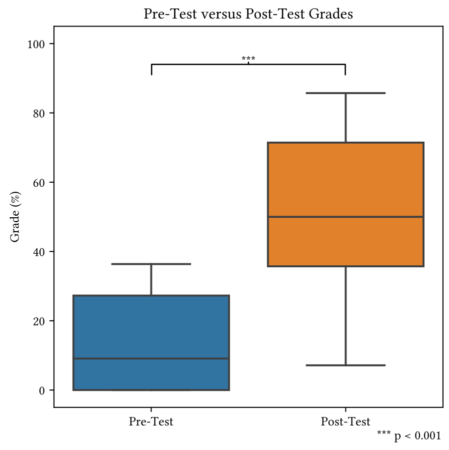


Fig. 2. Boxplots showing the differences between the pre- and post-test scores.

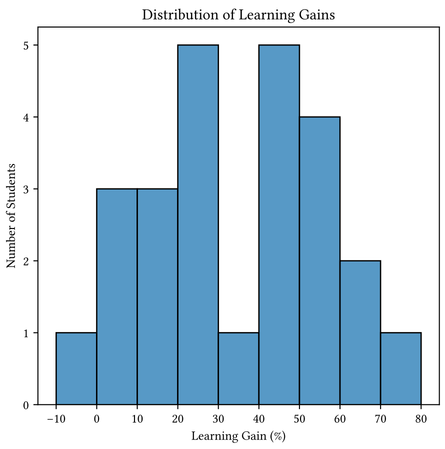
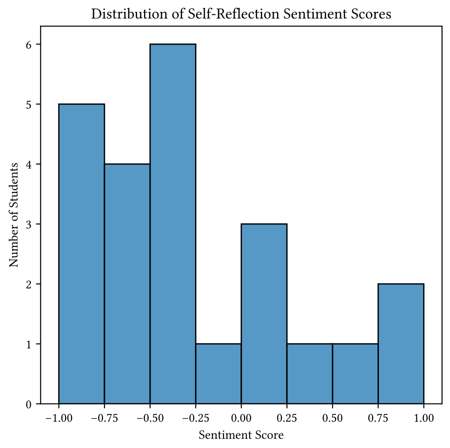


Fig. 3. Histogram depicting the distribution of learning gains achieved.

34 to 553 characters in length ($\bar{x} = 189.1, \tilde{x} = 151.0, s = 131.7$). Of the 23 students who provided a self-reflection, 2 students did so without identifying the specific issues they found problematic, while 17 students specifically reported issues they had missed. For students who mentioned specific issues, they reported a mean of $\bar{x} = 59.6\%$ ($\tilde{x} = 55.8\%, s = 26.5\%, x_{\text{min}} = 11.1\%, x_{\text{max}} = 100.0\%$) of issues they had missed. A sentiment analysis of these comments revealed that they had a negative sentiment on average ($\bar{x} = -0.278, \tilde{x} = -0.340, s = 0.546$). Fig. 4 shows the distribution of the sentiment scores and Fig. 5 a plot of these sentiment scores against their corresponding student's learning gain. A simple linear regression was performed to test if learning gains were an adequate predictor of sentiment scores ($R^2 = 0.115, F(1, 21) = 2.733, p = 0.113$), but the results were not significant.

4.3 Feedback

In addition to a self-reflection, 22 students provided feedback in the form of responses answering an open-ended question about their overall satisfaction with the lesson. Comments ranged in length from 4 to 924 characters ($\bar{x} = 177.3, \tilde{x} = 132.5, s = 188.4$) and the sentiment analysis performed on these comments yielded a positive mean sentiment of $\bar{x} = 0.556$ ($\tilde{x} = 0.668$, $s = 0.374$). Fig. 6 shows the distribution of these sentiment scores and Fig. 7 the relationship between sentiment scores and learning gains. Once more, we performed a simple linear regression to test if learning gains were an adequate predictor of the sentiment scores, yielding results that were not significant ($R^2 = 0.165, F(1, 19) = 3.767, p = 0.0673$). Finally, using line-by-line coding, we identified prominent themes that were present in the feedback. On the one hand, 15 students highlighted the positive aspects of the lesson, describing it as "clear", "good", and



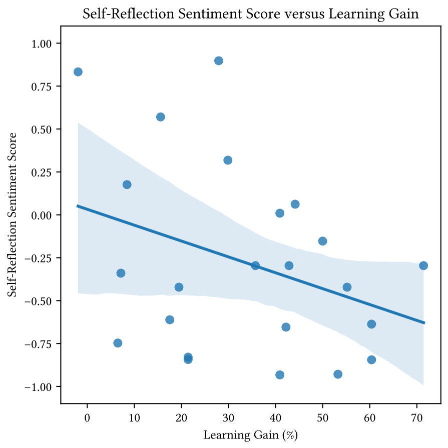


Fig. 4. While the distribution of students' self reflection sentiment scores was wide, it shows a tendency towards negative scores.

Fig. 5. There is a negative relationship between learning gains and self-reflection sentiment scores, but learning gains did not significantly predict sentiment scores.

"useful". This is best summarized by one particular student, who described the lesson as follows: "*[The lesson is] very pedagogical, [it] gives a good first overview without being too dense or discouraging, [and] makes you want to learn more by yourself!*"[7]. On the other hand, 3 students pointed out their lack of experience in the chosen programming language as a limitation, while 3 different students found the lesson difficult.

5 Discussion

The results of our study suggest that using a code review notebook to support an online lesson on software engineering education leads to tangible short-term learning gains. The fact that—on average—students detected 34% more code quality issues in the post-test than in the pre-test indicates that the lesson had a considerable impact on students' understanding of what constitutes good quality JavaScript code and on their ability to detect and annotate issues present in the code snippet. These results suggest that our online lesson format could serve to address one of the issues identified by Indriasari *et al.* [12], namely, that students often lack the skills necessary to perform code reviews [24]. Specifically, our lesson format could be used to address issues similar to the one experienced by Stalhane *et al.*, who reported that almost all students found only 2-3 out of 20-30 defects (10%), even though they were expected to find around 60% [22].

[7] "Très pédagogique, donne un bon premier aperçu sans pour autant être trop dense et décourageant, donne envie d'aller apprendre plus par soi-même !" (Translated from French by the authors.)

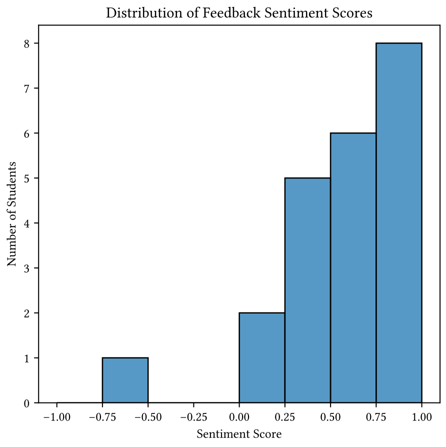


Fig. 6. The distribution of sentiment scores for the feedback provided was predominantly positive.

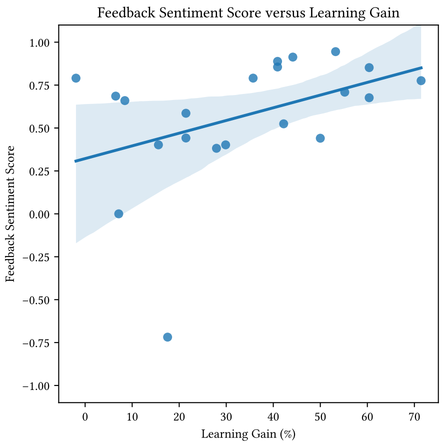


Fig. 7. While there is a slightly positive relationship between learning gains and feedback sentiment scores, learning gains did not significantly predict sentiment scores.

According to the authors, one of the possible explanations for this gap could be that the “process was unfamiliar and hence daunting” [22] for students. Thus, a code review notebook similar to the one used in this study could be exploited to guide students through the code review process before exposing them to more complicated code review exercises. Our work also sheds light on how to help address some of the concerns teachers have with the peer code review process. In a study of two computer science university-level courses, Kubincová *et al.* noted that teachers perceived the feedback provided by students (to other students) as focusing mostly on minor coding issues [14]. By harnessing a code review notebook similar to the one used in our experiment, teachers could provide specific examples of both the type of issues that they want students to look out for, as well as the way they want students to highlight these issues when they encounter them while reviewing their peers’ code.

Our analysis of the self-reflection comments provides evidence that students took the time to reflect on the learning experience. Exercises promoting self-reflection have been shown to be positively correlated with student success in software engineering courses [17] and students have reported finding self-reflection helpful [16]. Instructors could harness answers from these exercises in conjunction with the performance on code reviews to identify learning gaps, and thus personalize teaching strategies targeting students struggling with the concepts at hand. Furthermore, while the sentiment analysis of these self-reflection comments yielded a negative score on average, this result could be due to the way the question was posed to students. More concretely, the fact that we specifically asked students to reflect on the issues they had not been able to identify could have negatively biased the sentiment scores of their responses.

Conversely, the feedback provided was overall positive, indicating that the lesson content and the code review notebook format were predominantly well-received by students. Specific themes and comments that emerged from our qualitative analysis highlight the potential that code review notebooks have to serve as an introduction to complicated or broad topics that students can then go on to explore on their own. As there are a vast number of issues that can be encountered when performing an open-ended code review exercise, using a code review notebook to motivate the most common ones could be useful for instructors looking to prepare students for open-ended peer code review exercises.

Finally, the regression analysis performed shows that learning gains were not a good predictor of either the sentiment score of students' self-reflection comments or students' feedback on the lesson. While this study's sample size is limited, the results of our regression analysis could suggest that actual performance on the code review exercises was independent of motivation to reflect on the learning experience and of the perception of the lesson itself. This makes these types of lessons more adequate as a first exposure to the code review process for novice programmers, who should not feel discouraged if they do not perform as well as expected.

6 Limitations and Future Work

Our study has some limitations worth highlighting. First, as previously mentioned, while the sample size achieved in this study is suitable for an initial within-subjects short-term learning gains analysis, data from more subjects could allow us to consolidate our findings and draw clearer conclusions with respect to how learning gains are related to self-reflection and feedback. Second, our learning activity took place outside the formal classroom, as an optional exercise. This might have resulted in sample bias, as those students who responded to our request for participation might have been students already motivated enough to take part in an ungraded exercise. To address this, the experiment should be reproduced in the context of a formal software engineering course, possibly as a mandatory—albeit ungraded—exercise, which would provide us with a possibly less-biased cohort of students. Third, this study focuses on one particular use of code review notebooks, which concerns introducing code linting concepts for JavaScript programming. In order to generalize our conclusions, it is necessary to replicate our methodology with code review notebooks tackling other subjects—such as introducing basic programming concepts (e.g., loops, conditional statements, functions)—that could also be applicable to software engineering education. Finally, while feedback on the lesson was mostly positive, some negative comments regarding the usability of the interface emerged during our analysis. In future work, we aim to address these limitations and explore the potential to expand the code review notebook to more use cases.

7 Conclusion

In this paper, we presented results from a case study aimed at assessing the impact of code review notebooks on students' learning experiences in the context of an online lesson on detecting code quality issues in JavaScript. Results from our mixed-method analysis provide insight into the potential for code review notebooks to help students learn how to perform code reviews. Our findings are particularly pertinent to practitioners who use or are looking to incorporate code reviews into their courses. By introducing the code review process using an online lesson following the code review notebook format, instructors could prepare students to conduct reviews that are both useful for themselves and for the student receiving the feedback. As more instructors adopt such approaches, research into how best to scaffold these online lessons for different audiences (e.g., age groups, fields of study) could shed more light on the impact our approach could have in domains outside of software engineering education.

References

1. Airbnb: Airbnb JavaScript Style Guide (2012). URL airbnb.io/javascript/
2. Anewalt, K.: Using Peer Review as a Vehicle for Communication Skill Development and Active Learning. Journal of Computing Sciences in Colleges **21**(2), 148–155 (2005)
3. Ardıç, B., Yurdakul, İ., Tüzün, E.: Creation of a Serious Game for Teaching Code Review: An Experience Report. In: Proceedings of the 2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T), pp. 204–208. IEEE, New York, NY, USA (2020). DOI 10.1109/CSEET49119.2020.9206173
4. Bergin, J.: Fourteen Pedagogical Patterns. In: M. Devos, A. Rüping (eds.) EuroPLoP 2000. Universitaetsverlag Konstanz (2000)
5. Brooke, J.: SUS: A 'Quick and Dirty' Usability Scale. In: Usability Evaluation In Industry. CRC Press, London, UK (1996)
6. Charmaz, K.: Constructing Grounded Theory: A Practical Guide through Qualitative Analysis. Sage, London, UK (2006)
7. Farah, J.C., Soares Machado, J., Torres da Cunha, P., Ingram, S., Gillet, D.: An End-to-End Data Pipeline for Managing Learning Analytics. In: 2021 19th International Conference on Information Technology Based Higher Education and Training (ITHET). IEEE, New York, NY, USA (2021). DOI 10.1109/ITHET50392.2021.9759783
8. Farah, J.C., Spaenlehauer, B., Rodríguez-Triana, M.J., Ingram, S., Gillet, D.: Toward Code Review Notebooks. In: 2022 International Conference on Advanced Learning Technologies (ICALT), pp. 209–211. IEEE, New York, NY, USA (2022). DOI 10.1109/ICALT55010.2022.00068
9. Gillet, D., Vonèche-Cardia, I., Farah, J.C., Phan Hoang, K.L., Rodríguez-Triana, M.J.: Integrated Model for Comprehensive Digital Education Platforms. In: 2022 IEEE Global Engineering Education Conference (EDUCON), pp. 1586–1592. IEEE, New York, NY, USA (2022). DOI 10.1109/EDUCON52537.2022.9766795
10. Hsing, C., Gennarelli, V.: Using GitHub in the Classroom Predicts Student Learning Outcomes and Classroom Experiences: Findings from a Survey of Students

and Teachers. In: Proceedings of the 50th ACM Technical Symposium on Computer Science Education, pp. 672–678. ACM, New York, NY, USA (2019). DOI 10.1145/3287324.3287460

11. Hutto, C.J., Gilbert, E.: VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text. In: Proceedings of the Eighth International AAAI Conference on Weblogs and Social Media, pp. 216–225. AAAI, Palo Alto, CA, USA (2014)
12. Indriasari, T.D., Luxton-Reilly, A., Denny, P.: A Review of Peer Code Review in Higher Education. ACM Transactions on Computing Education **20**(3) (2020). DOI 10.1145/3403935
13. Johnson, S.C.: Lint, A C Program Checker (1978)
14. Kubincová, Z., Homola, M.: Code Review in Computer Science Courses: Take One. In: H. Xie, E. Popescu, G. Hancke, B. Fernández Manjón (eds.) Advances in Web-Based Learning – ICWL 2017, *Lecture Notes in Computer Science*, vol. 10473, pp. 125–135. Springer, Cham, Switzerland (2017). DOI 10.1007/978-3-319-66733-1_14
15. Li, X., Prasad, C.: Effectively Teaching Coding Standards in Programming. In: Proceedings of the 6th Conference on Information Technology Education (SIGITE '05), pp. 239–244. ACM, New York, NY, USA (2005). DOI 10.1145/1095714.1095770
16. Minocha, S., Thomas, P.G.: Collaborative Learning in a Wiki Environment: Experiences from a Software Engineering Course. New Review of Hypermedia and Multimedia **13**(2), 187–209 (2007). DOI 10.1080/13614560701712667
17. Pedrosa, D., Fontes, M.M., Araújo, T., Morais, C., Bettencourt, T., Pestana, P.D., Morgado, L., Cravino, J.: Metacognitive Challenges to Support Self-Reflection of Students in Online Software Engineering Education. In: 2021 4th International Conference of the Portuguese Society for Engineering Education (CISPEE). IEEE, New York, NY, USA (2021). DOI 10.1109/CISPEE47794.2021.9507230
18. Politz, J.G., Krishnamurthi, S., Fisler, K.: CaptainTeach: A Platform for In-Flow Peer Review of Programming Assignments. In: Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education (ITiCSE '14), p. 332. ACM, New York, NY, USA (2014). DOI 10.1145/2591708.2602687
19. Ribeiro Guimarães, J.P.: Serious Game for Learning Code Inspection Skills. Master's thesis, Universidade do Porto (2016)
20. Schrepp, M., Hinderks, A., Thomaschewski, J.: Design and Evaluation of a Short Version of the User Experience Questionnaire (UEQ-S). International Journal of Interactive Multimedia and Artificial Intelligence **4**(6), 103–108 (2017). DOI 10.9781/ijimai.2017.09.001
21. Song, X., Goldstein, S.C., Sakr, M.: Using Peer Code Review as an Educational Tool. In: Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education, pp. 173–179. ACM, New York, NY, USA (2020)
22. Stalhane, T., Kutay, C., Al-Kilidar, H., Jeffery, R.: Teaching the Process of Code Review. In: Proceedings of the 2004 Australian Software Engineering Conference (ASWEC '04), pp. 271–278. IEEE, New York, NY, USA (2004)
23. Tang, M.: Caesar: A Social Code Review Tool for Programming Education. Master's thesis, Massachusetts Institute of Technology (2011)
24. Trytten, D.A.: A Design for Team Peer Code Review. ACM SIGCSE Bulletin **37**(1), 455–459 (2005). DOI 10.1145/1047124.1047492
25. Zakas, N.C.: ESLint (2013). URL eslint.org
26. Zong, Z., Wang, Y., Schunn, C.D.: Why Students Want to Provide Feedback to their Peers: Drivers of Feedback Quantity and Variation by Type of Course. Journal of Psychology in Africa **31**(4), 336–343 (2021)