WIP: Code Insight: Combining Code Reading and Debugging Practices for Active Learning in Entry-Level Computer Science Courses

Keerti Banweer
School of Computer Science
University of Oklahoma
Norman, USA
keerti.banweer@ou.edu

Deborah A. Trytten
School of Computer Science
University of Oklahoma
Norman, USA
dtrytten@ou.edu

***Abstract*—This work-in-progress innovative practice paper describes an active learning in-class activity to be used in introductory programming courses. In the software development industry, critical thinking, problem-solving, code review, and debugging are vital skill sets software engineers require to build software that meets requirements. Commercial software companies often have extensive code review processes to make sure that developed code is of high quality. Many new developers work in quality assurance and maintenance of software products. For these jobs, learning to understand code written by peers is an essential skill.**

Code review skills are rarely explicitly taught in required programming classes in CS, especially at the entry-level. In beginning programming classes, students are usually thrilled if they can find one way to make a program work. They rarely see, let alone generate, multiple approaches to solving a single problem. In this research, we propose an active learning approach that provides students with experience reviewing and analyzing existing code. In our activities, students practice code review skills by exploring multiple approaches to solve a single problem.

We have developed an in-class active learning approach, called Code Insight, for entry-level programming courses. Code Insight activities are designed based on student-created answers to previous homework and midterm problems. Students are given the code to upload into their development environment, so they have the tools available that a typical software developer would have. Students use a classroom response system to report the correctness of these solutions by responding to multiple-choice questions. Preliminary results show that students prefer doing this activity to listening to lectures. In addition, students report that they learn more programming from this structure of the exercise. The degree to which Code Insight can improve student learning in beginning programming will be assessed using performance on other learning assignments, particularly those on programming questions in the midterm and final examinations.

***Index Terms*—Software engineering, Programming, Qualitative, Thematic analysis, Debugging**

I. Introduction

Students in entry-level programming classes can struggle with writing good-quality code [8] and find it difficult to debug programs [9]. In the introduction to programming classes, beginner programmers face many challenges while learning to program. Challenges include misconceptions, syntactical errors in code, or conceptual knowledge of programming. Innovative practices can help students address the difficulties learning programming concepts [35]. Code-writing skills have been found to be directly related to code-reading and code-tracing skills among novice programmers. Beginner programmers who are able to effectively review and explain a written code are found to perform better in their program writing skills [32], [36]. Therefore, our purpose in this study is to explore and analyze code reading activities that would encourage students to review existing code and improve their code writing and analyzing skills. Computer programming classes often focus on students developing code starting with an empty sheet of paper, which does not provide them with opportunities to review existing code or evaluate and explore different approaches to a single programming problem. While this experience is valuable, it does not align with the software industry, where most people work on developing and maintaining existing code. One industry-based practice that is rarely used in CS education is code review.

Code review is performed when programmers have their code read, critiqued, and approved by another programmer. Code review is an essential practice in mature software development organizations. Code review and debugging are critical skill sets required by the software industry. Code review practice is commonly used by developers in the software industry, such as Google, to improve the quality of the code, maintain consistency in formatting and documentation, ensure adequate testing, and improve security. A motivation for reviewing code is to encourage developers to write code that is easily read and understood, which is essential to software maintenance. Code review is one way that software organizations teach new developers how to write code that aligns with the organization's practices [1]. A survey on Open Source Software (OSS) projects found that developers could spend close to 6 hours per week on average reviewing peers' code or preparing code for a peer's review. Code review practices are commonly included in organizational protocols that software developers have to follow [3]. At Microsoft, for example, approximately 50,000 developers practice code review in a given month. Large software development companies have developed tools that

are used internally to automate code review like Microsoft's CodeFlow [2].

We propose an in-class active learning approach called **Code Insight** for entry-level programming courses. Our goal in designing this activity is to allow students to review more examples of code for each programming concept. Through this activity, we encourage students to review and understand written code and answer the multiple-choice question based on their analysis.

The proposed Code Insight activity is designed to utilize the benefits of an in-class active learning approach. Our aim is to address challenges of common misconceptions of concepts and lack of code reading skills among beginner programmers with no additional workload on the students and a reasonable workload on the teaching staff.

II. Literature Review

Student peer review has been implemented and explored in higher education in multiple academic disciplines for many years, where students review their peers' work after submitting individual work and provide assessments of their peer's work either in a group setting or individually [15]. Student peer review is found to be beneficial in providing timely and meaningful feedback to the students on their work, having meaningful discussions for an improved learning experience, and reducing the instructor's workload [16] [17]. Code review can be a special type of peer review. Code review has been found to benefit students with the development of critical thinking skills and improved learning experience by discovering different programming concepts from their peers. However, introducing code review activities in the course module throughout the semester has been found to add extra workload for both the student and the teaching staff, although online tools may reduce the burden [11].

Code review practices have been implemented in CS courses and utilize various approaches, such as individually submitted reviews, collaborative work on a review, and a combination of individual and collaborative code review [7]. Previous work on code review used an exercise where students reviewed code written by peer(s) for a project they had already completed [5], [6]. Previous work on code review by individual students has used both in-class activity or as a take-home assignment [4], [13], [17], [23], [25], [26], [27], [39]. Students were required to submit their own work on a programming problem and then provide a review of a peer's code using a rubric provided by the instructors. Many code review implementations used a predefined rubric or review criteria to assist students in reviewing the code and provide feedback with a given checklist of best programming practices [19], [20], [21], [22]. In some studies, students were allowed to reflect on the reviews of their work and resubmit the updated solution by utilizing feedback from their peers [13], [17], [23], [25], [39]. Receiving timely feedback and being given a chance to improve their original work has been found to be beneficial to learning. When a code review activity is implemented in an upper-level CS capstone course, it was found that students also developed professional skills and generated higher-quality code by addressing feedback from their peers [18].

Some of the code review implementations utilize online tools for students to review peer's code and provide their review and suggestions on the written code [19], [20], [23], [25], [26]. Online tools provide students with structured review entries on programming assignments using a predefined rubric. Some previous work used a collaborative code review approach in entry-level programming classes [5], [12], [20]. In a collaborative approach, the students work on a project in a team of two or more students and then review and submit feedback on their peer's work in a group. Code review can be performed using an online tool or face-to-face. Hundhausen et al. compared the results from online and face-to-face implementations of a code review activity. They found that face-to-face implementations result in better students' attitudes towards self-efficacy and an overall positive experience of the code review exercise [24].

Performing code review of peer's work is a time-consuming process, resulting in some rushed reviews [25]. For novice programmers, it is sometimes challenging to understand advanced code with complex programming concepts [26]. For individual code reviews to be meaningfully graded, course instructors or teaching assistants would have to carefully read and critique each student's code [5]. This may not be possible in larger classes where programs are often evaluated using code testing harnesses. In addition, while using a fixed rubric provides students with criteria for assessing code quality, it can direct students to examine superficial elements of the code (following code conventions, indentation, etc.) instead of looking deeply at the more important programming choices that have been made (e.g. nested loops versus single loops, unnecessarily complicated or disorganized logic) [4], [5], [20]. In addition, time-consuming exercises mean that peer code review can be done a limited number of times during the semester, limiting the strategy's impact on student learning.

An activity like code review needs an exercise structure that is easy to implement, creates challenging but doable learning experiences for students, and can be performed frequently in class to allow students to gain the maximum benefit.

In this work, we discuss how to design and implement code reading exercises for students in entry-level programming courses. We will discuss the following research question:

- RQ. In what ways can code review activities be structured to help students learn introductory programming concepts?

The design for Code Insight is motivated by the team peer code review implementation [12]. We utilize the approach of parallel code review [14], where students will see different approaches to solving the same problem.

III. Code Insight

Code Insight is designed as an in-class learning activity integrated into each programming concept and implemented after the end of each concept discussion in an introduction to the programming class. Our focus is to introduce more

examples for each topic in the programming course and allow students to see and analyze multiple approaches to solving a given programming problem. Students will evaluate existing code using a set of multiple-choice options to direct them in reviewing key areas of the code and understand the logic implemented to solve a problem. By providing more examples on a single topic and discussing the functionality of the written code improves student learning [28]. Worked examples on programming problems have been found to develop cognitive skills and increase student understanding of the topic [29]. We used multiple-choice questions in the Code Insight activity, which includes distractors with different misconceptions common among novice programmers because it has been shown that analyzing working examples can be crucial in gaining problem-solving skills [30]. Code reading questions have been shown to provide an improved learning experience for students when provided with feedback on the given code, and programming misconceptions are discussed [33]. Active learning has generally been shown to increase student engagement and improve the learning experience during class [34].

A. Developing Questions

We used the previous semester's exam questions and student answers to develop the multiple-choice questions. An advantage of this approach is that it leverages faculty time that has already been spent developing concise and properly scoped questions. Some students' answers in the exam provide interesting approaches (in addition to the approach the instructor intended) to solving a problem, while others can show common student misunderstandings. The key to making the process of identifying student answers efficient is to leverage the grading process to identify both common misconceptions and unexpected approaches. This is done with Gradescope software, which provides editable rubric-based grading [37], although it could also be done using post-it notes for people who grade directly from papers. Common misconceptions can be identified based on which rubric items produce the most point deductions. In addition, we include a rubric item (with no point deduction or bonus) for interesting solutions. This item is used to identify student answers that are unique, either in that they provide an unexpected correct solution to the problem or show a common misunderstanding. While this tag is visible to students, no student has ever asked what it means.

Student solutions are typed into an Integrated Development Environment (IDE) to produce programs students can download from the course management system. We edit the code to increase or decrease the difficulty of the Code Insight problem, and to avoid plagiarizing from students. For example, mistakes irrelevant to the class topic are fixed, code is simplified to be as short as possible, short identifiers are replaced with meaningful ones, and code indentation and structure are edited to improve readability. Assumptions that are relevant to the question are also developed if necessary. A typical assumption is that users enter input in the correct format. We then write multiple-choice items to direct students to examine specific parts of the code carefully. We have used two approaches to writing multiple-choice items:

- ask students if the code works all the time, never, or sometimes. This option makes items quick to write.
- draw the student's attention to specific lines of code that may have problems. These options can be harder to write.

Typically, two separate code examples are created for each problem. All examples should be compiled successfully.

```
public static void main(String[] args) {
    Scanner keyboard = new Scanner(System.in);
    final int PAGES_PER_REAM = 500;
    int numOfpages = keyboard.nextInt();
    keyboard.next(); keyboard.next();
        keyboard.next();
    String personsName = keyboard.nextLine();
    int numberOfReams =
        (int)Math.ceil(numOfpages /
        (double)PAGES_PER_REAM);
    System.out.println(personsName + ": " +
        numberOfReams + " ream(s)");
}
```

The above code is an example of Code Insight activity. This problem asked the student to read in user input from the console in this format "*number-of-pages* pages printed by *user-name*" and print out "*user-name*: *number-of-reams* ream(s)". Specific numbers and names would replace data in italics. Students have to convert the number of pages of paper to the number of reams, which requires both a floating point division and a ceiling operator. The multiple choice question is:

Which one of the following is true about the given section of code?

a) Code will correctly print the expected output
b) double cast of PAGES_PER_REAM is not needed
c) int cast for the Math.ceil method is not needed
d) keyboard.next() should not be called 3 times

In this case, the code is printing out the expected output and choice a) is the correct answer. The distractors represent common misconceptions. The double cast is needed to assure that the division has a floating point result and the int cast is needed because the Math.ceil() method in Java unexpectedly returns a double value, not an integer. The keyboard.next() method needs to be called three times to skip over the words "pages printed by".

B. Classroom Implementation

After completing each primary programming language concept, a Code Insight activity is implemented in the classroom. These activities are typically used at the end of each of the ten topical units of the course. The process is summarized in Fig. 1 and described below the figure.

The code is published when the activity begins. The problem is explained to students. The original wording of the problem from the prior examination is used. The example code, previously downloaded to the professor's IDE, is run to show students the format and expectations for input. Students then download the example code from the course management

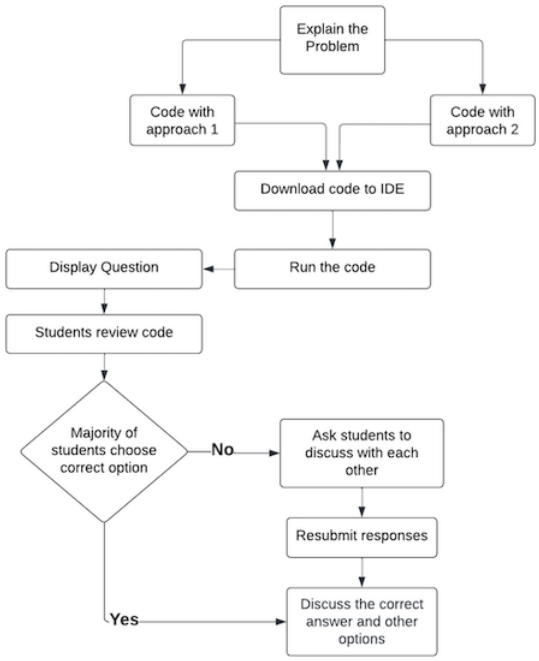

Fig. 1. Implementation Process

system to their laptops to evaluate the code. The input/output assumptions for the problem are given to the students and the multiple choice question is displayed in the classroom.

The students select the best option using the classroom response system. If the majority of students get the correct answer, the instructor briefly explains the answer and moves on to the next example. Otherwise, student responses indicate that there are still substantial misconceptions. We ask students to talk with adjacent student(s) about their responses and encourage them to seek out those with different answers. These discussions typically take five additional minutes. After these discussions, students provide a second response to the question. The responses usually move towards a single option, which is usually the correct solution. The solution is then discussed briefly, although longer discussions may be necessary if student responses converge on an incorrect solution. A pair of exercises typically takes between 10 and 15 minutes, although more time is necessary as the programs get longer, and if there are many students with given misconceptions.

IV. Methodology

Code Insight was implemented during the Spring 2024 semester in the introduction to programming classes. This course is taught in two different sections, one designed for students with some prior programming experience and another for students with no programming experience. This design was inspired by Cohoon's separation of students without prior programming experience into a dedicated section [38]. The courses have identical topical coverage, with the course for non-programmers having additional laboratory time to allow students to perform pair programming for projects. Either course fulfills the prerequisite for the second programming course. Our Institutional Review Board (IRB) approved this study.

We will use *quantitative methods* to analyze how students' performance on programming questions in the exams (midterms and final) compares to that of students who took the class before the implementation of Code Insight. The structure of the exams and assignments was not changed, although the individual questions were modified to keep students from memorizing answers. We will utilize a common grading rubric to analyze the performance (overall number of points awarded) and the particular errors (whether students make different mistakes, as reflected in individual rubric items). To effectively analyze the impact of code insight activities on the students' learning experience, we will map the exam question on a specific topic with the respective code insight activity. This will allow us to directly compare the student's response on the code insight activity with the performance on the programming question in the exam. We expect that students will make different errors on programs that they write on the examinations. We will review the students' responses for code review and the applied rubrics in the exam questions across each topic. This will allow us to evaluate the impact of Code Insight activity on student learning.

For the *qualitative data*, we used an anonymous online and in-person surveys to collect open-ended feedback from the students. Early in the semester, we emailed these surveys to students resulting, predictably, in unacceptably small numbers of responses. Later in the semester, we had participants complete the surveys during class time by following the steps approved by IRB. We received 17 responses to the online survey, showing promising preliminary results. Eighty percent of the students neither found this activity too confusing, nor too easy. Eighty percent of the students also agreed that the activity helped them understand the class concepts. More than half of the students responded that they would prefer Code Insight over a lecture. Most importantly, more than ninety percent of the students reported feeling more confident in their debugging and code review skills. While the number of responses remains small, we consider the results encouraging.

V. Future work

In our future implementation, we will integrate Code Insight with a specific team-based learning strategy developed by Michaelson and Fink [31]. In this strategy, students first answer the questions independently and then answer the same questions with a strategically chosen small group. The small group answers are made visible to the whole class and used as a basis for a student-led whole-class discussion ending with a consensus on the correct answer. The key to making this format of team-based learning successful is the development of very hard multiple-choice questions. We believe that the Code Insight activity is capable of providing these exercises and assisting novice programmers with improving their code-writing skills.

REFERENCES

[1] Sadowski, C., Söderberg, E., Church, L., Sipko, M. and Bacchelli, A., 2018, May. "Modern code review: a case study at google." In Proceedings of the 40th international conference on software engineering: Software engineering in practice (pp. 181-190).

[2] Bosu, A., Greiler, M., & Bird, C. (2015, May). "Characteristics of useful code reviews: An empirical study at Microsoft." In 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories (pp. 146-156). IEEE.

[3] Bosu, A., & Carver, J. C. (2013, October). "Impact of peer code review on peer impression formation: A survey." In 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (pp. 133-142). IEEE.

[4] Turner, S.A., Pérez-Quiñones, M.A. and Edwards, S.H., 2018. "Peer review in CS2: Conceptual learning and high-level thinking." ACM Transactions on Computing Education (TOCE), 18(3), pp.1-37.

[5] Hundhausen, C. D., Agrawal, A., & Agarwal, P. (2013). "Talking about code: Integrating pedagogical code reviews into early computing courses." ACM Transactions on Computing Education (TOCE), 13(3), 1-28.

[6] John Hamer, Helen Purchase, Andrew Luxton-Reilly, and Paul Denny. 2015. "A comparison of peer and tutor feedback." Assess. Eval. High. Educ. 40, 1 (2015), 151–164.

[7] Indriasari, T. D., Luxton-Reilly, A., & Denny, P. (2020). "A review of peer code review in higher education." ACM Transactions on Computing Education (TOCE), 20(3), 1-25.

[8] McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y.B.D., Laxer, C., Thomas, L., Utting, I. and Wilusz, T., 2001. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. In Working group reports from ITiCSE on Innovation and technology in computer science education (pp. 125-180).

[9] Lister, R., Adams, E.S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J.E., Sanders, K., Seppälä, O. and Simon, B., 2004. A multi-national study of reading and tracing skills in novice programmers. ACM SIGCSE Bulletin, 36(4), pp.119-150.

[10] McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L. and Zander, C., 2008. "Debugging: a review of the literature from an educational perspective." Computer Science Education, 18(2), pp.67-92.

[11] Song, X., Goldstein, S.C. and Sakr, M., 2020, June. "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).

[12] Trytten, D.A., 2005. "A design for team peer code review." ACM SIGCSE Bulletin, 37(1), pp.455-459.

[13] Carbonaro, A. and Ravaioli, M., 2017. "Peer assessment to promote deep learning and to reduce a gender gap in the traditional introductory programming course." Journal of e-Learning and Knowledge Society, 13(3).

[14] Luxton-Reilly, A., Lewis, A., & Plimmer, B. "Comparing sequential and parallel code review techniques for formative feedback." In Proceedings of the 20th Australasian Computing Education Conference, pp. 45-52, 2018.

[15] Huisman, B., Saab, N., van den Broek, P., & van Driel, J. (2019). The impact of formative peer feedback on higher education students' academic writing: a Meta-Analysis. Assessment & Evaluation in Higher Education, 44(6), 863–880. https://doi.org/10.1080/02602938.2018.1545896

[16] Søndergaard, H., & Mulder, R. A. (2012). Collaborative learning through formative peer review: pedagogy, programs and potential. Computer Science Education, 22(4), 343–367. https://doi.org/10.1080/08993408.2012.728041

[17] Wang, Y., Yijun, L.I., Collins, M. and Liu, P., 2008. Process improvement of peer code review and behavior analysis of its participants. ACM SIGCSE Bulletin, 40(1), pp.107-111.

[18] Krusche, S., Berisha, M. and Bruegge, B., 2016, May. Teaching code review management using branch based workflows. In Proceedings of the 38th International Conference on Software Engineering Companion (pp. 384-393).

[19] Hundhausen, C., Agrawal, A., Fairbrother, D. and Trevisan, M., 2010, March. Does studio-based instruction work in CS 1? An empirical comparison with a traditional approach. In Proceedings of the 41st ACM technical symposium on Computer science education (pp. 500-504).

[20] Hundhausen, C., Agrawal, A., Fairbrother, D. and Trevisan, M., 2009. Integrating pedagogical code reviews into a CS 1 course: an empirical study. ACM SIGCSE Bulletin, 41(1), pp.291-295.

[21] H. Hamalainen, J. Tarkkonen, K. Heikkinen, J. Ikonen and J. Porras, "Use of Peer-Review System for Enhancing Learning of Programming," 2009 Ninth IEEE International Conference on Advanced Learning Technologies, Riga, Latvia, 2009, pp. 658-660, doi: 10.1109/ICALT.2009.195.

[22] J. H. Hayes, I. R. Chemannoor, and E. A. Holbrook. 2011. Improved code defect detection with fault links. Softw. Test. Verif. Reliabil. 21, 4 (2011), 299–325. DOI:https://doi.org/10.1002/stvr.426

[23] Yanqing, W., Hang, L., Yanan, S., Yu, J. and Jie, Y., 2011, August. Learning outcomes of programming language courses based on peer code review model. In 2011 6th International Conference on Computer Science & Education (ICCSE) (pp. 751-754). IEEE.

[24] Hundhausen, C.D., Agarwal, P. and Trevisan, M., 2011, March. Online vs. face-to-face pedagogical code reviews: an empirical comparison. In Proceedings of the 42nd ACM technical symposium on Computer science education (pp. 117-122).

[25] Y. Wang, H. Li, Y. Feng, Y. Jiang, and Y. Liu. 2012. Assessment of programming language learning based on peer code review model: Implementation and experience report. Comput. Educ. 59, 2 (2012), 412–422. DOI: https://doi.org/10.1016/j.compedu.2012.01.007

[26] Y. Wang, Y. Liang, L. Liu, and Y. Liu. 2016. A multi-peer assessment platform for programming language learning: Considering group non-consensus and personal radicalness. Interact. Learn. Environ. 24, 8 (2016), 2011–2031. DOI:https://doi.org/10.1080/10494820.2015.1073748

[27] Jungkook Park, Yeong Hoon Park, Suin Kim, and Alice Oh. 2017. Eliph: Effective visualization of code history for peer assessment in programming education. In Proceedings of the 2017 ACMConference on Computer Supported Cooperative Work and Social Computing (CSCW'17). ACM, New York, NY, 458–467. DOI:https://doi.org/10.1145/2998181.2998285

[28] Pirolli, P. and Recker, M., 1994. Learning strategies and transfer in the domain of programming. Cognition and instruction, 12(3), pp.235-275.

[29] Pirolli, P., 1991. Effects of examples and their explanations in a lesson on recursion: A production system analysis. Cognition and Instruction, 8(3), pp.207-259.

[30] Trafton IV, J. G. (1994). The contributions of studying examples and solving problems to skill acquisition. Princeton University.

[31] Michaelsen, L.K., Knight, A.B. and Fink, L.D. eds., 2023. Team-based learning: A transformative use of small groups in college teaching. Taylor & Francis.

[32] Lister, R., Fidge, C. and Teague, D., 2009. Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. Acm sigcse bulletin, 41(3), pp.161-165.

[33] Sudol-DeLyser, L.A., Stehlik, M. and Carver, S., 2012, July. Code comprehension problems as learning events. In Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education (pp. 81-86).

[34] Yannier, N., Hudson, S.E., Koedinger, K.R., Hirsh-Pasek, K., Golinkoff, R.M., Munakata, Y., Doebel, S., Schwartz, D.L., Deslauriers, L., McCarty, L. and Callaghan, K., 2021. Active learning: "Hands-on" meets "minds-on". Science, 374(6563), pp.26-30.

[35] Qian, Y. and Lehman, J., 2017. Students' misconceptions and other difficulties in introductory programming: A literature review. ACM Transactions on Computing Education (TOCE), 18(1), pp.1-24.

[36] Lopez, M., Whalley, J., Robbins, P. and Lister, R., 2008, September. Relationships between reading, tracing and writing skills in introductory programming. In Proceedings of the fourth international workshop on computing education research (pp. 101-112).

[37] Singh, A., Karayev, S., Gutowski, K. and Abbeel, P., 2017, April. Gradescope: a fast, flexible, and fair system for scalable assessment of handwritten work. In Proceedings of the fourth (2017) acm conference on learning@ scale (pp. 81-88).

[38] Cohoon, J.P., 2007, March. An introductory course format for promoting diversity and retention. In Proceedings of the 38th SIGCSE technical symposium on Computer science education (pp. 395-399).

[39] Turner, S., Pérez-Quiñones, M.A., Edwards, S. and Chase, J., 2011, March. Student attitudes and motivation for peer review in CS2. In Proceedings of the 42nd ACM technical symposium on computer science education (pp. 347-352).