Improving Student Learning Through Required Exposure to Other Student Code Via Discussion Boards

Dr. Kyle D Feuz, Weber State Univesity

Kyle Feuz is an Associate Professor at Weber State University in the School of Computing. He earned his Ph.D from Washington State University under the guidance of Dr. Diane Cook in 2014. He also received his B.S and M.S in Computer Science from Utah State University in 2010 and 2011, respectively. He is actively engaged in research in Artificial Intelligence, Machine Learning, Computer Security and Computer Science education.

Dr. Linda DuHadway, Weber State University

Linda DuHadway has been in higher education for many years. She has degrees from Utah State University and received a PhD from the University of Utah with a focus in Computer Science Education. She is actively engaged in bringing a variety of innovative teaching methodologies into the traditional and online classroom. Currently her focus is implementing a new program called CS Flex. CS Flex offers a creative way to minimize the time barriers of higher education. It includes mastery learning, open entry, and flexible completion including support for accelerated progress.

Dr. Hugo Edilberto Valle, Weber State University

Dr. Hugo Valle is an Associate Professor of Computer Science at Weber State University. He received his Ph.D. in Physics in 2008 and MSc. in Physics in 2006, both from Vanderbilt University (Nashville, TN). His research interests are in IoT devices, Data Visualization, Software Development for particle detectors, sensors, microelectronics, and embedded systems. Previously, he was a member of the PHENIX collaboration at RHIC.

Dr. Richard C Fry, Weber State University

Richard Fry is a full professor at Weber State University in the School of Computing. He is actively involved in service-learning research and continues to partner with both local and global community organizations to engage Computer Science students in building sustainable Software Engineering projects. In 2014, his students took 2nd place in a national competition for their software solution supporting people with disabilities. Dr. Fry has also led more than 150 of his students to both Ghana and Thailand to work on Software Engineering projects affecting their global communities. He currently serves as the faculty in residence for the Center of Community Engaged Learning at Weber State University.

Prof. Kim Marie Murphy, Weber State University

Prof. Kim Murphy earned her M.S. degree from Utah State in 2010. Starting in 2018, Prof. Murphy was an Instructor at Weber State University in the department of Computer Science. From 1999 to 2018 she taught Computer Science classes at Weber High School.

Improving Student Learning Through Required Exposure to Other Student Code Via Discussion Boards

Abstract:

In a typical lower-division programming course, students rarely see other students' code outside of a paired programming exercise. This limits their exposure to the potentially powerful learning experience of seeing examples from other programmers. In this work, we explore the result of having required code sharing via discussion posts to increase a student's exposure to coding solutions, styles and practices (both good and bad). For each module in a Data Structures and Algorithms course, students post a small section of code, typically a single method or function, and also get to see the code posted from every other student in the class. They can choose to share a section of code that worked particularly well for them or submit code they are struggling with and want some help. The students are then required to respond to entries posted by other students. This creates a dialog between students and provides a mechanism for students to see how other students are coding a solution. The code students submit is from a low-stakes assignment. Students are allowed to see other student's submissions from the very beginning. They are not required to have completed the assignment or posted their own code to enter the discussion board. To identify the benefits of this assignment, the comments during one semester are analyzed and the results tracked over the course of the semester. The code posted is analyzed for a variety of quality markers such as variable names, commenting, syntax errors, logic errors, correctness, and handling of edge cases. The responses are analyzed for effective error corrections, alternative solutions provided, formatting changes recommended, etc. There is also an analysis of student expressions. Finally, and perhaps most interesting of all is how these details change over the course of the semester.

I. Introduction

Imposter syndrome has long been acknowledged within the Computer Science (CS) educational community with several studies reporting over 50% of CS oriented individuals exhibiting imposter syndrome [1]. Weber State University is an open-enrollment university. Students can enroll regardless of the current GPA, ACT and SAT scores or other common criteria used in a college admissions program. This means students in the classroom often come in at very different levels of preparation. Additionally, more than 50% of the student population is considered non-traditional. These are some of the factors that lead students to feel isolated and underprepared for pursuing a college degree, particularly in a STEM field.

In many computer science (CS) degree programs, a Data Structures and Algorithms class is especially challenging and is often a point where students get slowed in their degree progression [2]. This creates problems for retention and graduation rates. When students take this class, most of them are in their sophomore year. At this point in the degree, many students have not fully developed their problem solving and programming skills, and most of their coding feedback has come from the instructor. In some situations, students taking a traditional face-to-face course have the opportunity to do collaborative work. This can come in the form of pair programming, code sharing or some small group assignments. However, the majority of their work is individual assignments.

Evidence supports code-review, collaboration, pair programming and student-to-student interaction as valuable tools to improve learning and retention. These strategies are especially beneficial for under-represented populations [3,4]. The effectiveness of these activities varies depending on the course format. For example, in a face-to-face format, collaboration and pair programming works well. Some instructors have even been successful implementing distributed pair programming in an online course [5]. However, in a flexible schedule, online format (Flex), the implementation of these activities is particularly challenging. In the Flex format, students start the course at different points in the semester, work at their own pace and may not be working on the same module at the same time. The asynchronous nature of the class makes it particularly difficult for students to interact with each other. We propose the use of discussion boards within the learning management system to help create peer-to-peer code sharing experiences in a Flex class. In this study, we explore the benefits of using discussion boards for peer-to-peer code review. We believe an effective way to improve a student's confidence in their programming skills is to share their successes, ask for help with their struggles, and to review their peers' work. The students are required to post their code as part of the assignment. Students are also required to review or comment on their peers' posts.

Our goal is to show that code-sharing via online discussion boards can be used as an effective tool to help students improve their programming skills, be successful in the course and recognize that other students also struggle when working through the assignments.

The rest of this paper is organized as follows: the methods and techniques to analyze the data collected are discussed in Section II. Section III presents the results and findings. A discussion of the results is provided in Section IV. Related work is provided in Section V. Finally, our conclusions and plans for future work are presented in Section VI.

II. Methods

The Data Structures and Algorithms class is a computer science course taught in the C++ language. The course is composed of seven different modules and each of the modules cover a different set of topics. The amount of content and the time allocated to each module is similar across all modules. Certain modules and topics tend to be easier for students to grasp than other modules. Table 1 lists the core topics of each module. In each case, students are asked to implement some or all of the functionality of the data structure or algorithm as well as use their implementation in solving a specific programming problem.

The class is offered in an online, flexible format. Our goal is to allow students to move through the modules at their own pace. When a student completes one module they are able to immediately move on to the next module in the course. Modules can be completed early, on-time, or students can request an extension. Due dates for each module shift as extensions are requested. This flexibility means that students in the same section of the course will be working on different modules at any given point in time.

Table 1. Data Structures and Algorithms Module Topics

Module	**Topics**
Module 1	C++ classes, Pointers, Dynamic Memory, Operator Overloading
Module 2	Copy-constructor, Assignment operator, Destructor, Move constructor, Move assignment operator, Inheritance
Module 3	Linked Lists, Templates
Module 4	Recursion, Stacks, Queues
Module 5	Sorting, Search, Hashtables
Module 6	Trees, Binary Search Trees, Expression Trees, Balanced Trees
Module 7	Graphs, Dijstra's algorithm, Depth-first search, Breadth first search

Each module has an introductory programming assignment that includes a video walkthrough, a related try-it-out assignment, and a corresponding challenge assignment. As the students progress through the module, the responsibility for coding gets shifted from the instructor to the student. Until in the end, the student is coding the project on their own.

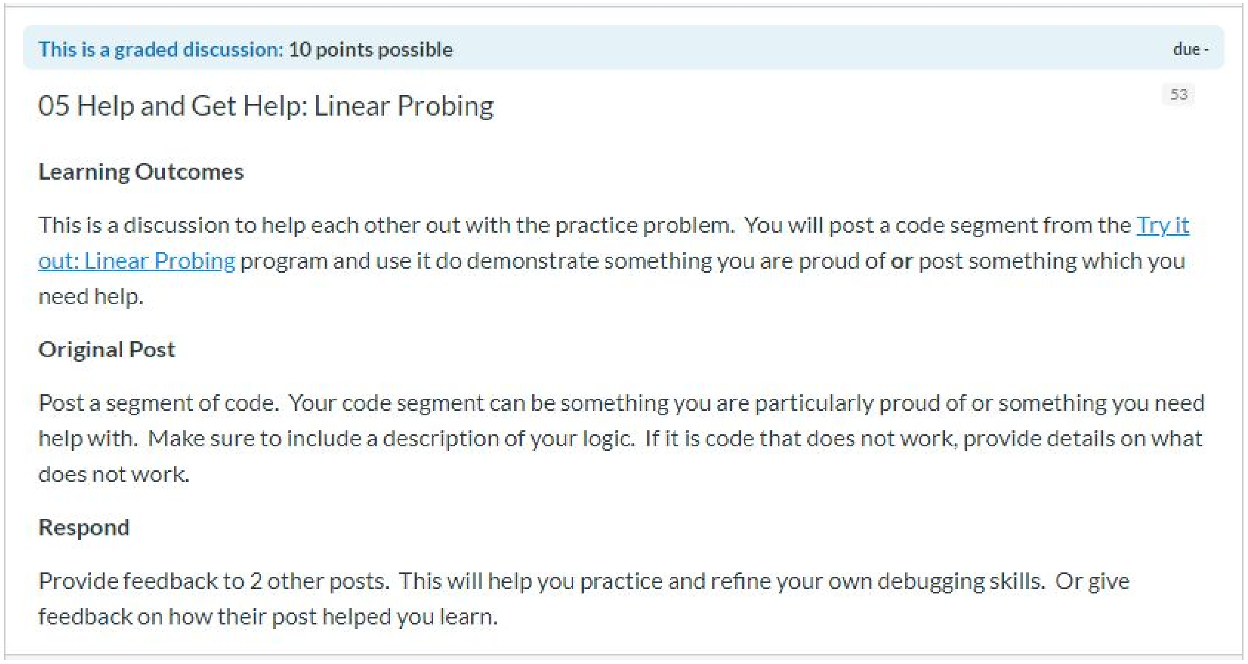
The walkthrough assignment is low-stakes and has the student follow along as the instructor writes some code. Most or all of the walkthrough assignments are demonstrated directly by the instructor. The student needs to listen, follow along, and write the code. The end program needs to compile correctly. As the students follow along they learn the new concepts and create a program at the same time.

The try-it-out assignment requires the student to go beyond what has been directly demonstrated. It uses similar concepts to the walkthrough assignment but applies them in a slightly different fashion. This is also a low-stakes assignment.

The challenge assignment is where the student is asked to demonstrate proficiency in the concepts being covered by the module. This assignment may use some code directly from the walkthrough or try-it-out assignments but it also asks students to go even further in demonstrating the application of the concepts being covered.

For each try-it-out assignment there is an associated discussion board with the same prompt for each module (except the assignment name). This prompt is shown in Figure 1. Students are encouraged to share their code openly and freely and to ask for help from other students. Code they get from the discussion board can be used directly in the try-it-out assignment. They do not have to post their own code, contribute to the discussion or submit the assignment before accessing the discussion board.

No special effort has been made to guide or moderate the discussion boards. Students received a copy of the syllabus which includes a statement about the classroom being an inclusive environment and prohibiting any discrimination or harassment. Each post has the student's name attached in a manner clearly visible to all students and the instructor.


This is a graded discussion: 10 points possible

due -

05 Help and Get Help: Linear Probing

53

Learning Outcomes

This is a discussion to help each other out with the practice problem. You will post a code segment from the Try it out: Linear Probing program and use it do demonstrate something you are proud of **or** post something which you need help.

Original Post

Post a segment of code. Your code segment can be something you are particularly proud of or something you need help with. Make sure to include a description of your logic. If it is code that does not work, provide details on what does not work.

Respond

Provide feedback to 2 other posts. This will help you practice and refine your own debugging skills. Or give feedback on how their post helped you learn.

Figure 1. Discussion Prompt

Students are given full-credit for the discussion if they make a valid attempt to post one code segment and respond to at least two other posts. For the try-it-out assignment they are given full credit if they make a valid attempt at implementing it. The solution does not have to be correct or even compile to receive credit. The student just needs to demonstrate that they tried something. The incentive for getting the code to work is not the grade. Instead, students know that they will face a similar problem on the challenge assignment so if they can get it working in the try-it-out assignment they will find the challenge assignment much easier to complete.

At the end of the semester the discussion board postings are analyzed to quantify how students interacted with the discussion board. Using a predetermined set of characteristics (see Table 2 and 3), two separate reviewers manually mark each post and reply as either having or not having the characteristic. The results are then compared for consistency and averaged together. The characteristics selected for analysis are chosen as a means of measuring whether or not the code-sharing supported the stated goals of improving students coding abilities, helping students be successful in the class and helping students see other students encountering the same challenges they face.

We also administer three exams throughout the semester. The first exam covers modules 1-3. The second exam covers modules 4 and 5 and the last exam covers modules 6 and 7. We compare the performance of students from this semester on the programming exam questions to the performance of students from previous semesters on similar programming exam questions as a means of quantitatively measuring the effectiveness of code-sharing in helping students learn to code and be successful in the class. The exam programming questions vary slightly between semesters so only modules 1, 2, 3, 4, and 6 have exam questions that are similar enough for a

Table 2. Student Post Characteristics

Description
1. Student includes code comments in the posted code snippet
2. Student uses descriptive variable names throughout the code snippet
3. The posted code snippet is free from any errors and functions correctly.
4. The posted code snippet contains one or more syntax errors
5. The posted code snippet contains one or more logic errors
6. The posted code snippet contains one or more logic errors that only occur under certain conditions (i.e. out-of-bounds access, etc)
7. The post contains additional thoughts, comments or extra information outside of the code snippet
8. The student expresses a sentiment that their code snippet is correct
9. The student expresses a lack of confidence that their code is correct
10. The student acknowledges that the code snippet may lack in efficiency, elegance, etc
11. The student ask direct question or uses a questioning phrase
12. The student uses a workaround to provide formatting and or syntax highlighting. (i.e. screenshot of code snippet, <pre> tags, etc)

Table 3. Student Reply Characteristics

Description
13. The student ask a question in their reply
14. The student answer a question in their reply
15. The student suggests an alternative solution to the code post.
16. The student expresses that they found the posted code snippet to be helpful
17. The student expresses that they found the posted code snippet to be unhelpful
18. The student expresses a positive sentiment about the code. (i.e Nice work, Good job, etc)
19. The student expresses that they had not considered using that approach before
20. The student expresses confusion about the code snippet
21. The student expresses a "me too" sentiment. (i.e. that they used a similar approach in their own code)
22. The student corrects a syntax error in the code snippet
23. The student corrects a logic error in the code snippet
24. The student references the efficiency (runtime, memory, etc) of the code snippet or suggested alternative
25. The student suggest a different choice of names for the variables used in the code snippet
26. The student suggests a change in the style of the code snippet. (formatting, whitespace, etc)
27. Other students reply to this reply either directly through a threaded reply or indirectly by reference this reply.

direct comparison to be meaningful. Furthermore, the questions for Module 1 and 2 have significant overlap and are combined and reported under Module 2.

III. Results

Over the course of the semester there were a total of 110 posts and 232 replies. Table 4. Shows the amount of participation in each module. After an initial drop in participation between the first and second module, the amount of participation was more consistent in the next few modules. Then we see another drop in participation towards the end of the course with modules six and seven having fewer posts and replies.

Table 4. Participation Statistics per Module

	Number of Participants	**Number of Posts**	**Number of Replies**
Module 1	20	19	36
Module 2	16	14	32
Module 3	17	17	33
Module 4	17	17	36
Module 5	16	16	37
Module 6	13	13	26
Module 7	14	14	32

The posts and replies were independently analyzed by two different reviewers and marked using the characteristics listed in Tables 2 and 3. The inter-rater reliability between reviewers on the post data using Pearson's Correlation Coefficient was .99 and the inter-rater reliability between reviews on the reply data was .91. The results have been averaged between the two reviewers and are listed in Tables 5 and 6.

Table 5 shows the percentage of student replies that demonstrate the given characteristic. We see a majority of replies expressing a positive sentiment. The next most commonly observed characteristics are expressing a helpful sentiment, and expressing a me too sentiment. On the other side of things, no reply suggested a change in variable names. Suggesting a style change almost never occurred. Similarly, we rarely observed an expression of unhelpfulness. We see a small percentage of replies that express confusion, correct an error, or answer a question.

Table 6 shows the percentage of student replies that demonstrate the given characteristic. We see a majority of replies expressing a positive sentiment. The next most commonly observed characteristics are expressing a helpful sentiment and expressing a me too sentiment. On the other side of things, no reply suggested a change in variable names. Suggesting a style change almost never occurred. Similarly, we rarely observed an expression of unhelpfulness. We see a small percentage of replies that express confusion, correct an error, or answer a question. Although small, we also see students asking additional questions, or expressing that they had not considered that approach.

Table 5. Percentage of Student Post Exhibiting Characteristic

Characteristic	**Percent of Posts**	**Characteristic**	**Percent of Posts**
1. Comments code	39.6%	7. Extra information	82.60%
2. Good variable names	88.33%	8. Says answer is correct	8.32%
3. Correct	74.27%	9. Says answer may not be correct	10.86%
4. Syntax errors	4.70%	10. Says answer has shortcomings	6.51%
5. Logic errors	23.28%	11. Asks a question	7.47%
6. Edge-case logic errors	9.55%	12. Workaround for formatting.	15.92%

Table 6. Percentage of Student Replies Exhibiting Characteristic

Characteristic	**Percent of Replies**	**Characteristic**	**Percent of Replies**
13. Asks a question	8.22%	21. Me too sentiment	23.89%
14. Answers a question	4.38%	22. Corrects a syntax error	0.83%
15. Alternative solution	28.48%	23. Corrects a logic error	4.33%
16. Helpful sentiment	34.60%	24. References efficiency	3.14%
17. Unhelpful sentiment	0.45%	25. Suggests name choice	0.00%
18. Positive sentiment	65.14%	26. Suggests style changes	0.22%
19. Did not think of that	7.71%	27. Generates response	6.84%
20. Expresses confusion	2.97%		

We also looked at how the prevalence of these characteristics changed over the course of the semester. For many characteristics the change over time was not significant or interesting. It would bounce back and forth around the average value. However, we have selected four characteristics from the posts and 6 characteristics from the replies that did appear to exhibit some interesting or noteworthy changes.

The trends exhibited in the characteristics of the student posts are shown in Figure 2. Students initially included comments in the code at nearly a 50% rate. By module 4 code comments peaked at nearly 65% of posts including comments in the code. From here things decline to less than 25% of students using comments in the code. We see a positive trend in the amount of posts that include correct code and a corresponding negative trend in the amount of code containing logic errors. The other characteristic which showed a positive trend was the workaround for formatting code. As one student would introduce a work-around for including syntax highlighting and maintaining whitespace formatting other students would copy the work around in subsequent modules.

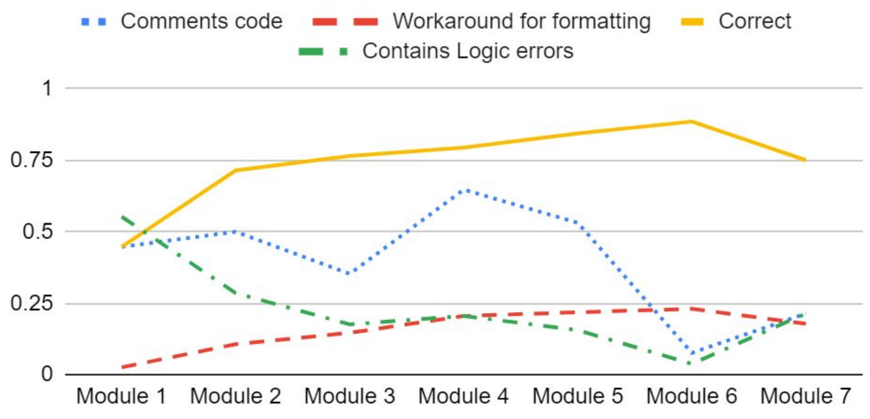


Figure 2. Student Trends in Posts

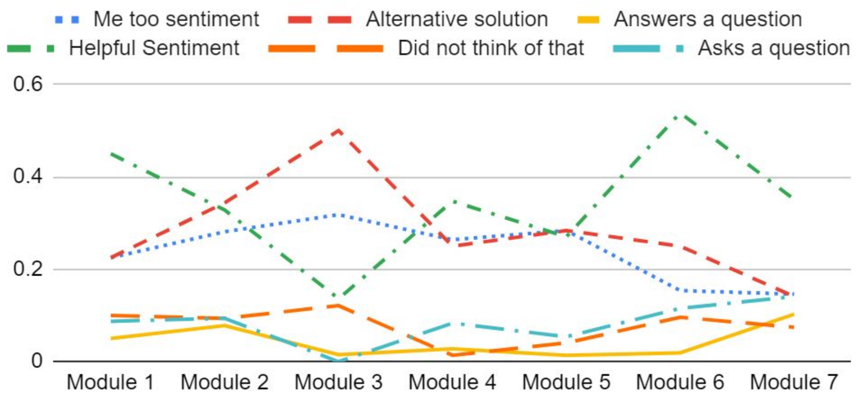


Figure 3. Student Trends in Replies

The trends exhibited in the student replies are shown in Figure 3. The replies expressing a me too sentiment have a downward trend. We see more students asking and answering questions in the early and later modules. The replies in the middle modules have fewer questions and answers. Students expressing a "did not think of that" sentiment is largely stable between modules with module 4 seeing the fewest replies expressing that sentiment and module 3 seeing the most replies expressing that sentiment.

We see an interesting pattern emerge with alternative solutions and helpful sentiments. Student replies that express the post was helpful are negatively correlated with student replies that suggest an alternative solution. It looks like when a student finds the post helpful they do not provide an alternative solution and when they find the post less helpful they instead suggest an alternative. The two characteristics together consistently make up about 70% of the replies.

It is important to realize that although each module was similar in the amount of content presented and that the amount of time students had to complete each module was identical, some module concepts were more challenging for students to grasp and implement then others. Anecdotally, we observed students struggle more with modules 1, 2, 6 and 7 then they did with modules 3,4 and 5. We see several of the trend lines reflecting this pattern.

To compare performance on the exam programming questions we looked at the percentage of students scoring 70% or higher on the programming exam questions that are similar across semesters. We compare this semester to the previous four semesters all taught by the same instructor. The results are shown in Figure 4. The percentage of students achieving a 70% or higher on the exam programming questions is similar or higher in the semester when students participated in the code-sharing discussion boards than in the other traditional semesters.

Although the results on the exam programming questions are promising, more work is needed to validate those improvements. Introducing code-sharing via discussion boards is not the only change that has taken place in this course compared to previous semester offerings. The course has also moved to an online flexible format. The use of code-sharing via discussion boards in other settings is currently being explored to isolate the effect of the discussion board.

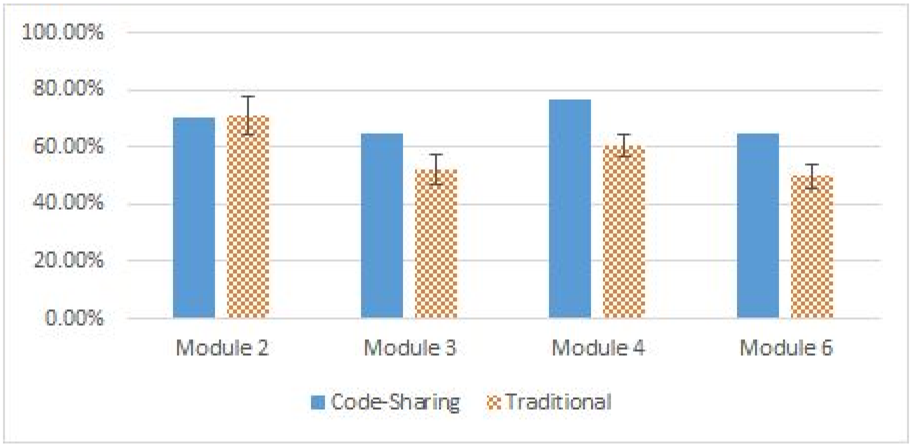


Figure 4. Percentage of Students Scoring at least 70% on the Exam Programming Questions

IV. Discussion

1. Good things we saw

We saw students helping students, students learning new ways of doing things, trying new ideas, answering questions, and having some back and forth dialog (max depth observed was 5). The number of correct postings improved over time. Students recommended other approaches and made suggestions.

The me too sentiment showed up often. There were students who expressed they solved a problem in a similar way. Others indicated that they struggled with the same issues. This opportunity to connect with others seems beneficial for many reasons. Often students feel isolated and that they are the only ones struggling. Seeing others are struggling as well helps them normalize the struggle inherent in learning new material and may reduce the effects of imposter syndrome.

Sharing successes can be beneficial to both the person that has succeeded and the one who hears of the success. Writing working code is inherently satisfying. Being able to tell someone else about it, increases the satisfaction. As other students see their classmate's success, they are more likely to anticipate their own success.

Logic errors are often difficult to see. If the testing is limited, students may not even realize they have included a logic error. One of the situations we saw was a student shared code and indicated that it was working. Then someone would reply and identify a concern about a particular logic error. They may have seen that the code had not protected against an edge case like when an array is empty and suggested that an if statement be added to handle that specific case. They may have noticed an off-by-one error where everything works great until it gets to the end of the array and then the loop writes something just outside the array bounds.

These errors are tricky to see. With multiple eyes on the code, it increases the chance that someone will notice. Once the error is identified, it can be easily fixed. Participating in these discussions helps the student who posted the code. They are able to fix it. It also helps those who are participating. Certainly the student who discovered the issue is rewarded for their ability to recognize the logic error. But even the student who just reads through the discussion benefits from hearing the conversation and can become more aware.

2. Things we saw that could be improved

Not every post had a reply. It would be nice if everyone had responses. There was a noticeable decline in the number of responses for the later posts. The first few posts had the most replies. By the end, some of the posts didn't have any replies and the others only had one or two.

Not every question was answered or even responded to. Questions were brought up in the replies. Often these questions went unanswered. They were valid questions and it would have been beneficial for the students to hear the answer.

This reduced response rate later in a thread or at the end of a discussion makes sense. Students participate in the discussion and move on. This may be an indication that fewer students are seeing the later posts and replies. One of the things we can investigate is how to encourage students to come back to the conversation after they have completed the required number of replies and moved on to another module.

3. Things we rarely saw

We saw few suggestions for stylistic improvements and no suggestions for better variable names. This may be a consequence of students not caring or considering it less of a priority.

Corrections of syntax errors were also rare. This is likely because there was less code that included syntax errors. Only 4.7% of code posted contained syntax errors. Compilers do a good job of pointing out syntax errors and students can often resolve them on their own.

4. Things we did not see

There were no negative or hostile comments. This was one of the best parts of the discussion. It was not clear at the onset that this would be the case. One of the concerns with a public discussion is you don't have control of what is said. There is a risk that someone will be disrespectful or unkind in their comments. The author of each post and reply was clearly visible which provided individual accountability for what was said. It was encouraging to see how helpful and kind the student remarks were. We hear negative things about 'students these days'. This is a chance to see the quality conversation they engaged in and say, 'Wow! Look at students these days'.

One of the most difficult things to say nicely is that something is wrong with someone else's code. The students were even able to do this. Consider this reply that identifies an error.

> “Wouldn’t the “index > size” need to be either “index >= size” or “index > size - 1” since if the index equals the size it would be out of bounds? Maybe I’m not thinking straight due to lack of sleep. Let me know what the consensus is.”

They also expressed appreciation for the corrections made.

> “This is great! I took your advice and didn't use the int variable. So far it has worked just fine. I always forget to add the null character until I look at your code.”

Having students participate in these conversations provides one more way to get students to engage in the code and topics of the course. It seems no matter how often we cover a topic, some students don’t hear or it just doesn’t make sense to them. Hearing the topics discussed by new voices and seeing code that has a different style, approaches a solution in a different way, or is organized differently broadens the student’s exposure to the topics of the course and to code and problem solving more generally.

V. Related Works

Research suggests that the benefits of working together on code include increased success rates in introductory courses, increased retention in the major, higher quality software, higher student confidence in solutions, and improvement in learning outcomes [6, 7, 8, 9]. While earlier works mainly address paired programming in a classroom setting, we concentrate on its impact on asynchronous collaboration via a discussion forum (in which students programmed individually and were expected to comment and receive feedback on different coding solutions, styles and practices) in an online, “self-paced” learning environment.

There are several similar studies that present findings on collaboration among novice programmers in non-traditional learning environments. Othman [10] describes a web-based system named Online Collaborative Learning System (OCLS) that has been developed to support collaboration and discussion for learning programming in a virtual environment. The "Think-Pair-Share" used in the study reflects the adoption of collaborative approaches in a virtual setting. Othman’s later study [11] also demonstrates a strong correlation between students’ logical thinking skills with their abilities to solve problems in an online collaborative environment.

Muller and Padberg [12] conducted two controlled experiments with 38 subjects on pair programming. They first studied the correlation between a pair’s feelgood factor and the pair’s implementation time and programming experience. In the second phase, rather than looking at the pairs, they focused on the individual’s programming experience and feelgood factor. The findings showed that a pair’s implementation time was uncorrelated to the pair’s programming experience, but there was a significant correlation with how comfortable the developers felt with paired programming during the session (the “feelgood” factor). It is our view that the prevalence of the me too sentiment expressed in the replies and the positive interactions between participants produces such a feelgood factor.

The most closely related work to our explorations into online student collaboration was done by Zacharis [13, 14]. These studies investigated the effectiveness of virtual pair programming (VPP) on student performance and satisfaction in an introductory Java course. The two groups consisted of virtually paired programming students and solo students. The two factors examined were code productivity and software quality. The results suggested that VPP is an effective pedagogical tool for flexible collaboration and an acceptable alternative to the individual/solo programming experience, regarding productivity, code quality, academic performance, and student satisfaction.

In our work, we more broadly explore how students collaborated and learned new ways of programming by making recommendations for improvements and evaluating suggestions received from others in an asynchronous manner.

VI. Conclusion

Allowing students the opportunity to see and interact with other's code is an important piece of learning and facilitates the writing of good software. This can be challenging in an online classroom environment and even more challenging when the students move through the course at their own pace. We have shown that using discussion boards is a feasible approach to provide students with this opportunity. Through the use of the code-sharing discussion boards we saw students relating to the experiences others shared, we saw students reporting that the discussion was helpful and we saw improvement on the exam programming questions.

We saw some items that can be improved going forward. Students often posed a question in the discussion board that went unanswered. We also saw some student posts that did not have a reply. This typically occurs towards the end of the discussion. Thinking about how to incentivize a student who may have already finished that module to go back and continue to be active on the discussion board could provide real benefits.

We plan to continue to use this approach in the Data Structures and Algorithms class and monitor the student experience. While we saw some indication that students performed better on the exam programming questions, further investigation is needed to determine if this is a result of the code sharing exercises or an artifact of other changes to the course structure. We also want to try this approach in additional classes to see if it is effective in a broad range of settings.

Overall, the experience was positive. We were especially pleased with how well the students communicated. They were respectful and thoughtful in their conversation. The students engaged in the discussion board. They learned new approaches and techniques they had not previously considered, they helped one another, encouraged each other and shared their code.

References

[1] A. Rosenstein, A. Raghu, and L. Porter. “Identifying the Prevalence of the Impostor Phenomenon Among Computer Science Students.” In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 2020. Portland, Oregon.

[2] R. Ball, L. Duhadway, K. Feuz, K., J. Jensen, B. Rague, and D. Weidman. “Applying Machine Learning to Improve Curriculum Design”. In *In SigCSE '19 (ACM Technical Symposium on Computer Science Education 2019)*. Minneapolis, Minnesota.

[3] L Barker, K. Garvin-Doxas, and E. Roberts E. “What can computer science learn from a fine arts approach to teaching?” *SIGCSE Bull*. vol 37, pp. 421–425, Feb. 2005. DOI:https://doi.org/10.1145/1047124.1047482

[4] L. Barker, K. Garvin-Doxas, and M. Jackson. “Defensive Climate in the Computer Science Classroom”. In *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education (SIGCSE ’02)*. Association for Computing Machinery, New York, NY, USA, pp. 43–47, 2002. DOI:https://doi.org/10.1145/563340.563354

[5] P. Baheti, E. Gehringer, and D. Stotts.. “Exploring the Efficacy of Distributed Pair Programming”. In: Wells D., Williams L. (eds) *Extreme Programming and Agile Methods - XP/Agile Universe* Lecture Notes in Computer Science, vol 2418, 2002. Springer, Berlin, Heidelberg.

[6] G. Braught, T. Wahls, and L.M. Eby. “The Case for Pair Programming in the Computer Science Classroom”. *TOCE*, 11, pp. 2:21, 2011.

[7] P. Maguire, R. Maguire, P. Hyland, and P. Marshall. Enhancing collaborative learning using pair programming: Who benefits?. All Ireland Journal of Higher Education, 6(2), 2014. https://ojs.aishe.org/index.php/aishe-j/article/view/141 [Accessed 1 February 2020].

[8] T. Van Toll III, R. Lee, and T. Ahlswede. "Evaluating the Usefulness of Pair Programming in a Classroom Setting," *6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007)*, Melbourne, Qld., 2007, pp. 302-308.
DOI: https://doi.org/10.1109/ICIS.2007.96

[9] L. Williams, E. Wiebe, K. Yang, M. Ferzli, and C. Miller. In Support of Pair Programming in the Introductory Computer Science Course, *Computer Science Education*, vol. 12:3, pp. 197-212, 2002. DOI: https://doi.org/10.1076/csed.12.3.197.8618

[10] M. Othman, F. Othman, and M. Hussain. "Designing Prototype Model of an Online Collaborative Learning System for Introductory Computer Programming Course, Procedia - Social and Behavioral Sciences 90, pp. 293-302, 2013.

[11] M. Othman, N. Zain.. “Online Collaboration for Programming: Assessing Students’ Cognitive Abilities. *Turkish Online Journal of Distance Education* , 16 (4) , 84-97, 2015. DOI: https://doi.org/10.17718/tojde.88618

[12] M. Muller and F. Padberg. “An empirical study about the feelgood factor in pair programming,” in *Proceedings of International Software Metrics Symposium*, pp. 151–158, Chicago, IL, USA, September 2004.

[13] N. Zacharis. “Evaluating the Effects of Virtual Pair Programming on Students’ Achievement and Satisfaction”. I*nternational Journal of Emerging Technologies in Learning*, vol. 4, pp. 34-39, 2009.

[14] N. Zacharis. "Measuring the Effects of Virtual Pair Programming in an Introductory Programming Java Course," in *IEEE Transactions on Education*, vol. 54, no. 1, pp. 168-170, 2011.