Paper ID #22992

A Comparison of Students Learning Programming with Online Modules, Instruction, and Team Activities

Dr. Jacqueline C. McNeil, University of Louisville

J.C. McNeil is an Assistant Professor for the Department of Engineering Fundamentals at University of Louisville. Research interests include First Year engineering, diversity in engineering, persistence, retention, co-op experiences, and longitudinal data. Contact email: j.mcneil@louisville.edu

Dr. Angela Thompson P.E., University of Louisville

Dr. Angela Thompson is an Assistant Professor in the Department of Engineering Fundamentals at the University of Louisville. Dr. Thompson received her PhD in Mechanical Engineering from the University of Louisville. Her research interests are in biomechanics and engineering education, particularly related to first-year programs and critical thinking instruction.

Nicholas Hawkins, University of Louisville

Nicholas Hawkins is a Graduate Teaching Assistance in the Engineering Fundamentals Department at the University of Louisville. A PhD student in Electrical and Computer Engineering, he received both his B.S. and M. Eng. from the University of Louisville in the same field. His research interests include power electronics and controls, as well as engineering education for first-year students.

A Comparison of Students Learning Programming with Online Modules, Instruction and Team Activities

Abstract

This paper investigates how first-year students learn programming through lectures involving team-based activities. Although programming instruction has traditionally been performed through individual means, advocates of "pair-programming" provide support for collaborative learning in software engineering. While these studies have explored dyads of programming students, this study goes further and investigates the effects of learning introductory computer programming in teams of four or five students. The primary research questions being investigated in this paper include: 1) how do team-based activities affect student participation?, 2) do team-based activities improve learning outcomes on programming assignments?, and 3) did student-reported effort, time, and value of programming change between the two years? To answer these questions, data was collected from an introductory engineering course contributing five weeks to programming instruction. Two sets of data have been collected: the initial set from an entirely individual, module version of the curriculum (Fall 2016), and the second set from a curriculum with added lectures and team-based activities (Fall 2017). Homework performance data were collected from the set of problems common to both years, along with the results of a survey taken by students about their perceptions of the programming portion of the course.

Statistical analysis revealed no significant difference in overall homework scores between the two years, but there was a significant increase in scores on programming projects, which were a set of more challenging problems at the end of Chapters 2-5 homework assignments. Average number of attempts per problem and number of problems completed did not change significantly between the two years. Results of the survey show that students generally perceived a lower workload and felt that the programming material was more valuable to them with in-class lectures and team activities.

Future direction based on this study indicate the potential need for more in-class instruction, either in the form of more team activities or lectures. Pair programming, dyads, has been shown to be successful in the literature and will be considered in this course in the future.

Introduction

Collaborative learning is an educational tool that utilizes the idea that students learn better together than alone. Students learning in pairs or groups tend to discuss ideas as they learn them, hearing the topic from different perspectives, and retain the information for longer periods of time. This sentiment has been applied to various academic disciplines, including software engineering education.

This paper addresses programming instruction in an introductory engineering course. The course was newly implemented in Fall 2016; introduction to programming was one of several topics covered in this course. Instructors received numerous accounts of negative feedback related to programming following the first iteration in 2016. Much of the negative feedback was in regard to the online textbook, which was among the primary tools for student learning in 2016. Nearly half of the students felt that the programming instruction was “not at all valuable”. Due to the motivation to improve upon the instruction for this course, concepts involving collaborative learning were employed to both improve student learning and increase enjoyment of the programming content.

Literature Review

Paired programming is a commonly used technique in academic settings throughout the country, and stems from a training methodology in the software industry known as Extreme Programming, or XP. XP was developed in 1996 by the programming industry to respond quickly to rapidly changing customer demands [1]. It is comprised of multiple practices designed to decrease time-to-market for programming solutions and was among the first major developments to introduce pair programming. Several universities have considered the use of XP in an academic setting, though there is some concern as to whether it truly applies. XP is primarily for increased efficiency in code development and does not necessarily include an improvement in skill proficiencies.

However, pair programming has been found to provide significant improvements to teaching software in the past. Many previous examinations of pair programming have shown that it improves program quality and assignment scores [2] [3] [4] [5], and others have shown that there is a decrease in the amount of time needed to complete those assignments [2], though the amount of time that differs is not consistent across these studies. Other investigations have concluded that students who program in pairs are at least more likely to pass a given assignment [6].

Students have also felt better about programming courses because of pair programming. Many students have noted that they enjoy the courses more due to the use of pair programming [7] [8] [9] [10], and it has increased student confidence in programming [11]. Multiple studies have found that the use of pair programming has increased the likelihood that a student stays in a course [12] [13] [14], or even stays a computer science major [12] [13] [15].

Research Questions

Considering these developments in the use of pair programming within an educational setting, the instructors considered teaching programming in a team setting. Would teaching programming to students with lectures and team activities have any effect on student performance and student attitudes in the course? The research questions addressed in this paper are:

1) How do team-based activities with programming topics affect student participation?

2) How do team-based activities affect learning outcomes on programming assignments?

3) Did student self-reported effort, time, and value of programming change with efforts to add active learning and in-class instruction?

Methods

Engineering Methods, Tools, and Practice I (ENGR 110) is a large introductory course. In the Fall 2016 course, there were 643 students enrolled in 18 sections, and in the Fall 2017 course there were 609 students. This course is required for engineering students of all engineering disciplines within the University. Most students take the course during their freshman year. This course is taught by 2 faculty and 4 graduate teaching assistants (TAs). The course consists of many topics related to introductory engineering concepts, such as an introduction to each discipline, critical thinking, teamwork, communication, and a variety of engineering tools including Microsoft Excel, hand-drawing of engineering graphics, and Python programming.

Table 1 shows demographics and distribution of majors for the ENGR 110 course in Fall 2016 and Fall 2017:

Table 1: demographics for ENGR 110 in Fall 2016 and Fall 2017.

	Fall 2016	**Fall 2017**
Gender		
Male	503	458
Female	137	137
Race		
Non-Resident Alien	16	13
Black	34	28
Asian	43	35
Hispanic	23	34
White	495	449
Race Unknown	0	1
Two or More Races	29	35
Major		
Bioengineering	77	75
Civil Engineering	55	63
Computer Engineering & Computer Science	129	124
Chemical Engineering	75	78
Electrical Engineering	49	58
Industrial Engineering	25	23
Mechanical Engineering	169	141
Pre-Engineering	3	1
Non-Engineering	5	4
Undeclared	53	28
TOTAL	***640***	***595***

Programming Instruction and Assignments

The programming section of ENGR 110 is comprised of 5-6 weeks, each week with a set of homework problems that include 1-3 programming projects. Homework sets were completed in Pearson's MyProgrammingLab (MPL), an online programming platform that provides feedback on incorrect code entries. The homework problems generally require fewer than 5 lines of code to answer a homework problem relating to a single programming concept (e.g. define a variable, write an expression comparing two variables, write an *if* statement), whereas the projects ask the student to write a program that accomplishes a specific task requiring blocks of code that can be up to 30 lines long.

The comparison being made takes account of two separate semesters of the course, Fall 2016 and Fall 2017. In 2016, the programming component of the course was a 6-week portion of the course with Chapters 1-6. For this study, the researchers only used Chapters 1-5 to be consistent with the chapters used in 2017. In Fall 2016, the programming and graphics components of the course occurred during the same weeks, during which graphics work was completed mostly in-class and programming was mostly done outside of class. There were no lectures on programming except for an initial introductory lecture on how to use the online system, MyProgrammingLabs. In Fall 2017, programming was completed before graphics was started, though other course activities took place at the same time as programming, and lectures (30-50 minutes) were provided in class that pertained to the Chapter homework problems and projects. Additionally, the 6th unit (and weekly homework assignment) on programming was not included in Fall 2017. Chapter 6 primarily covered how to program with external files, which was thought to be a narrow topic for general engineering course instruction, and was not covered in Fall 2017. There was a change in workload due to the change in when graphics was covered and dropping Chapter 6 during Fall 2017, but is outside the scope of this study.

As an additional alteration to the 2017 course, each class was separated into 4-5-person teams, which participated in a team activity in class prior to the due date of each programming project. In teams, the students were given the prompt for one of the Chapter projects and tasked with drawing a flowchart for the code that would solve the problem (they were not, however, tasked with writing the code during class, but were expected to write and submit the code individually for the project by the homework due date). The purpose of this was to use a team environment to help students think through the logic of the problem and visualize the code collaboratively.

Data Analysis

To determine whether the addition of programming lectures and team activities in 2017 improved student learning, student performance on homework assignments were assessed and an end-of-semester survey was given to assess student perceptions about programming and their learning.

Performance data were reviewed for a subset of students (3 of 18 sections in both Fall 2016 and Fall 2017), as these sections were taught by the same TA both years. The total pools of students are similar in size ($N_{2016} = 95$ and $N_{2017} = 100$). Of the programming problems assigned for homework each year, 91 of those problems were in both years' curriculum. From these results, three metrics were analyzed: average scores (number of problems correct out of the 91 assigned), number of attempted problems (out of the 91 assigned), and average number of attempts per problem.

Statistical analysis was performed using Minitab on two sets of homework data: the entire set of 91 homework problems common to both years, and the subset of those correlating to the programming projects only. Two-sided t-tests were performed to see if the sets of data between years had a statistically significant difference using a 0.05 significance level.

SPSS was used to calculate the chi-squared results relating to survey responses. The data used in the analysis was collected from surveys in 2016 and 2017, which are written in detail below. The data in both 2016 and 2017 was normal and was tested at a significance level of 0.05.

Another metric of comparison were responses from a survey. The survey asked for students' feedback regarding three course components: Excel, programming, and graphics. The survey was given upon the completion of all components near the end of the semester both years. In 2016, the survey was deployed when programming and graphics components had just finished. However, in 2017, programming was moved to the middle of the semester. The end of programming and the survey were separated by the graphics component, which consisted of 5 weeks. Thus, for the 2017 survey, students had 5 weeks before taking the survey at the end of the semester.

The results of this survey look at the entire population of the course from both years, as opposed to the three-section subset from which the scores were taken. The questions related to programming that were asked in both years' surveys are listed below:

- How would you rate the amount of programming work required? (Answer options: Not enough, Fair amount, Too much)
- How much time did you spend (on average) doing programming work outside of class? (Answer options: Less than 1 hour, 2-4 hours, 5-7 hours, 8-10 hours, Over 10 hours)
- How valuable do you feel the programming was to you? (Answer options: Very Valuable, Somewhat valuable, Not at all)

The purpose of the questions below was to inquire about student perceptions on the individual course resources that were available to them for programming. Students had the opportunity to report what they *enjoyed* and what they thought helped them *learn*. These additional questions regarding programming were asked only in the 2017 survey, which read:

- Which of the following did you enjoy in programming?
- Which of the following helped you learn in programming?

The answer options for these questions were the following:

- Lectures / Slides
- Team Activities / Flow Charts
- MyProgrammingLab Homework
- Textbook
- Other Resources (online, friends, etc.) What other resources were used?

If "other resources" was selected, students were able to write in other resources used.

Finally, an open-ended question was asked at the end of each survey that was written as follows:

- 2016: Anything else that you found valuable or think we could do differently in ENGR 110?
- 2017: What could we do different in ENGR 110 to make it better for next year?

Results

Below are the results from the Chapter homework in MyProgrammingLabs (Table 2). For all three metrics, statistical tests concluded that there are no significant differences, as none of the p-values meet the significance level of 0.05 (Table 2).

Full Homework Dataset

Table 2: T-test results for 91 homework problems assigned in both 2016 and 2017.

Quantity	**Mean (Std. Dev.)**		**P-Value**
Year	*2016*	*2017*	
Score (out of 91)	86.7 (9.84)	84.3 (15.4)	0.217
Problems Attempted (out of 91)	87.9 (9.08)	85.3 (15.0)	0.150
Average Attempts per Problem	2.46 (1.03)	2.64 (2.45)	0.514

Programming Project Analysis

Table 3 shows results for the 7 programming projects that were at the end of the homework in Chapters 2-5 that were common to both years. These problems were weighted higher (5 points each) compared to other homework problems which were worth 1 point each.

Table 3: T-test results for 7 "Programming Projects" common to both 2016 and 2017.

Quantity	**Mean (Std. Dev.)**		**P-Value**
Year	*2016*	*2017*	
Score (out of 7)	5.61 (2.17)	6.17 (1.53)	0.039
Problems Attempted (out of 7)	6.18 (1.73)	6.55 (1.27)	0.091
Average Attempts per Problem	6.99 (7.20)	7.11 (6.57)	0.905

These results look specifically at the projects, as those were significantly more in-depth programming questions that may be a better indicator of student learning. In this analysis, there is a statistically significant difference in scores between 2016 and 2017, where the mean in 2017 is higher.

Survey Results

Figure 1 shows that student perception of workload was fairer in 2017 than in 2016. Students who felt that the amount of work required for the programming was “too much” dropped substantially (16%). A chi-squared test was performed to examine if there was a significant difference between 2016 and 2017 fair workload. There was a significant difference between students’ reports of fair workload in 2016 and 2017, $\chi2$ (6, N = 804) = 22.85, p = 0.001.

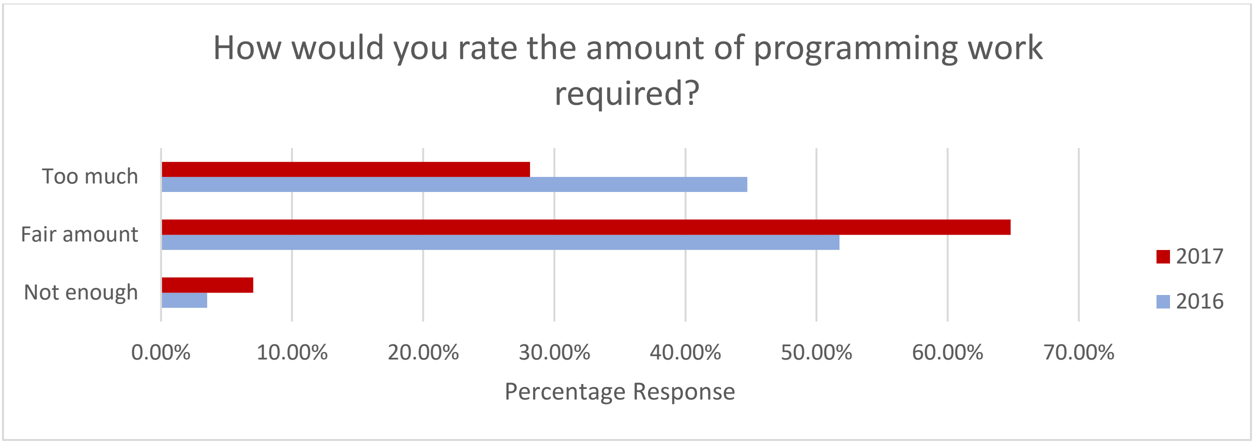

Figure 1: Survey results regarding student perceptions of workload.

Figure 2 shows the amount of time students were spending on programming assignments (2-4 hours was intended). Notably, the percentage of students responding "More than 10 hours per week" dropped from approximately 5% to nearly 0. Students claiming more than four hours per week decreased in 2017 compared to 2016 (25.45% of respondents versus 37.94%, respectively). A chi-squared test was performed to examine if there was a significant difference between time spent on programming in 2016 and 2017. There was not a significant difference between students' reports of time spent on programming in 2016 and 2017, $\chi 2$ (20, N = 804) = 22.06, $p > 0.05$, $p = 0.337$.

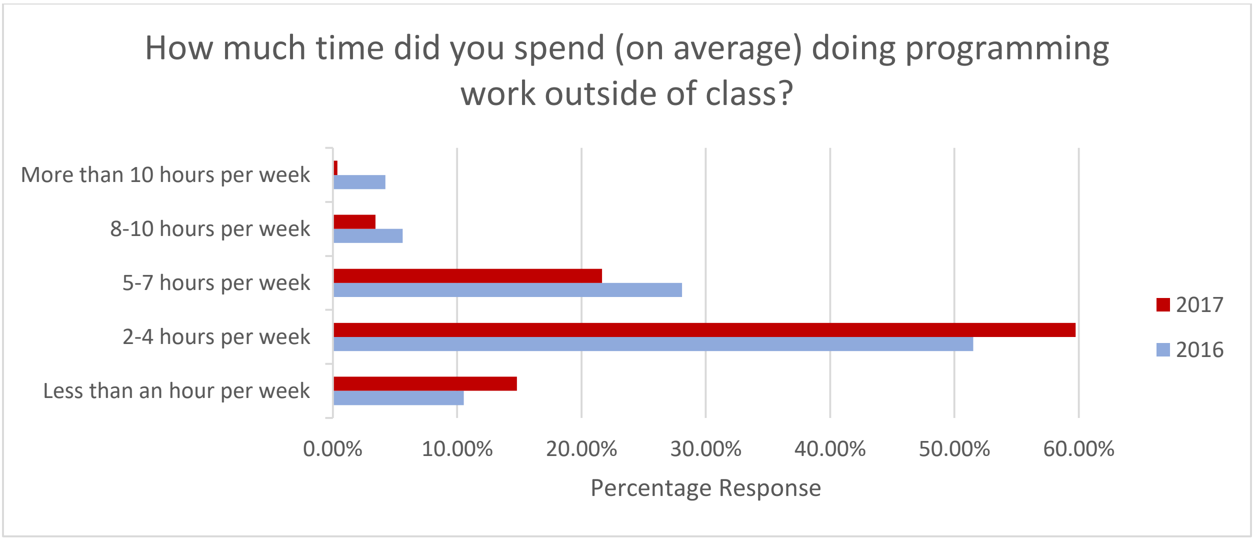

Figure 2: Survey results regarding student perceptions on time requirements for programming.

Figure 3 shows that in 2016 there were more students who saw no value in the programming than students who either saw some to a lot of value. In 2017, however, "Somewhat valuable" overtakes the other two categories. A chi-squared test was performed to examine if there was a significant difference between 2016 and 2017 value. There was a significant difference between students' reports of value in 2016 and 2017, $\chi 2$ (6, N = 804) = 20.45, $p = 0.002$.

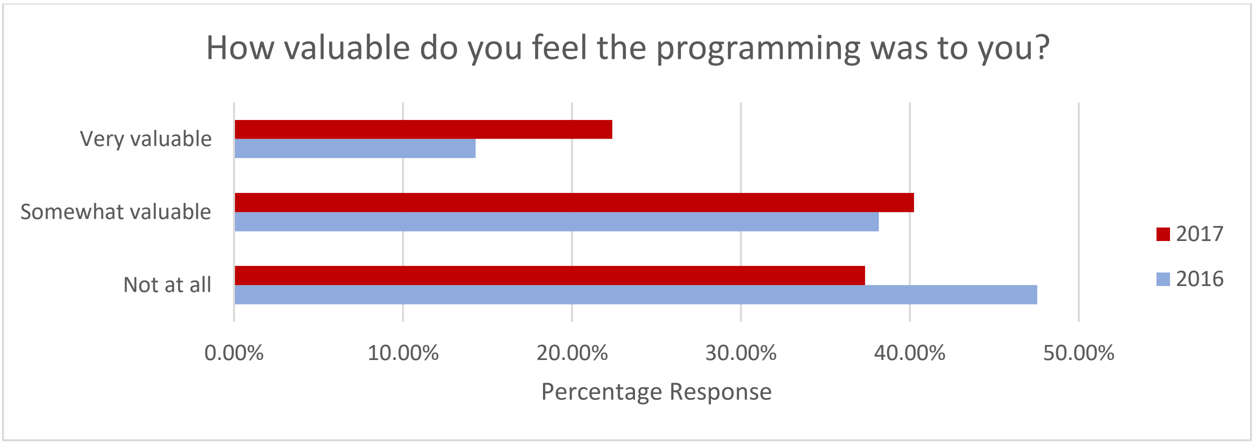

Figure 3: Survey results for student perceptions on programming value.

In Figure 4, 56.8% of students said that they enjoyed the team activities either *some* or *a lot*, and 53.3% of students said the same regarding the in-class lectures. Notably, the textbook was perceived to be both unenjoyable and unhelpful for learning. Figure 5 suggests that students found the lectures more helpful than the team activities; 36.5% of students found the team activities to not be helpful at all, while only 29.2% said the same of the lectures. MyProgrammingLab homework assignments were seen by students as most helpful for their learning. A large percentage of students found other resources helpful. These included primarily friends and internet sources such as YouTube and Code Academy.

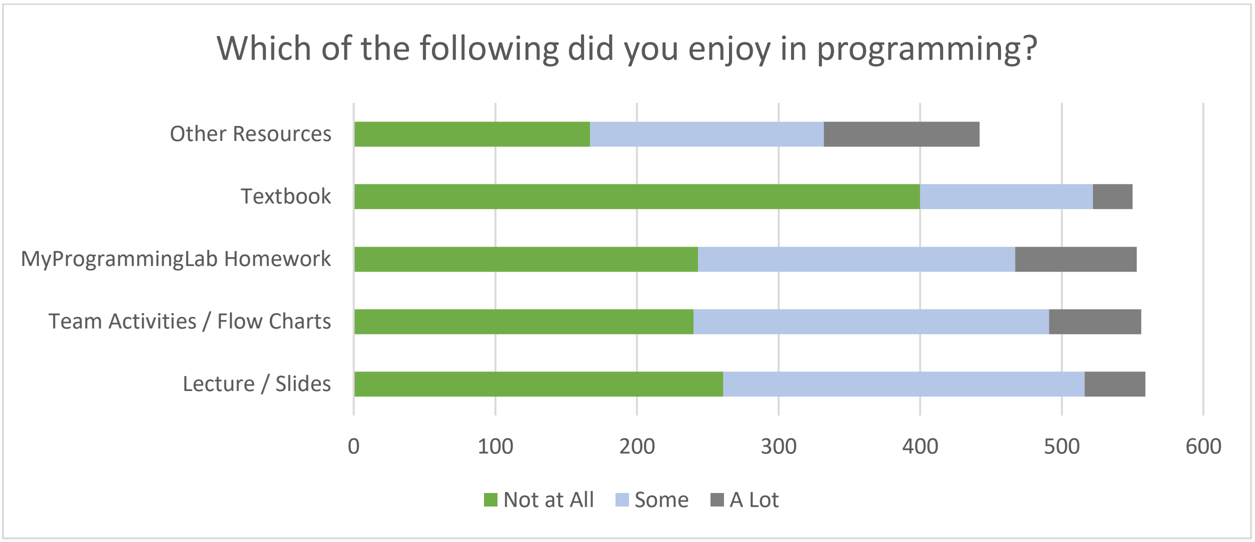

Figure 4: Survey results in 2017 on student enjoyment in programming.

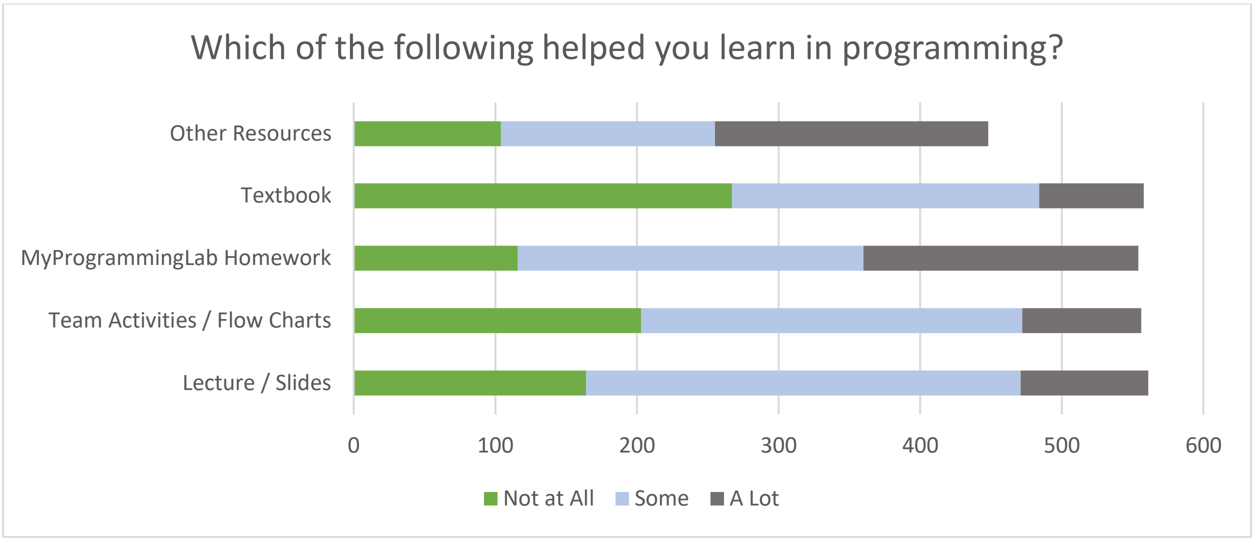

Figure 5: Survey results in 2017 on student learning in programming.

Finally, each year's survey had an open-ended response question at the end of the survey that was not specific to programming but asked for general feedback about the course. Table 4 outlines student interest in the subject of programming based on the responses to this question.

Table 4: Summary of open-ended survey question response rate.

Year	**Students Enrolled**	**Number of Open-Ended Question Responses**	**Responses that Relate to Programming**
2016	640	184 (28.8%)	118 (64.1% of responses)
2017	595	484 (81.3%)	206 (42.6% of responses)

While 2016 had a much lower percentage of responses in general, the rate of responses that discussed programming was much higher in 2016 than in 2017. Note that this method includes responses that cite either "programming", "python", or "coding" but does not account for misspellings. Almost all of the responses were negative regarding programming (felt programming content was poor or insufficient) and only one response in each year that was categorized as positive. Positive comments were determined by phrases such as 'liked programming' and 'programming was good.'

Table 5 below is a summary of the survey responses that relate to programming (referred to as just "responses"). Note that each response is not mutually exclusive to one category (i.e. one student response can be placed in more than one category).

Table 5: Summary of open-ended question responses relating to programming.

Theme	**Percent responses 2016**	**Percent responses 2017**
Need for more/better in-class instruction	57 (51.8%)	112 (69.6%)
Poor course resources (eText/MPL)	58 (52.7%)	32 (19.9%)
Too much material (quantity or depth)	11 (10.0%)	27 (16.8%)
Material not relevant enough	7 (6.4%)	8 (5.0%)
Remove programming entirely	0 (0.0%)	11 (6.8%)
Other	13 (11.8%)	27 (16.8%)
Total	*133 (112.7%)*	*217 (105.3%)*

In 2016, there was not in-class instruction, so responses relating to the "need for more/better instruction" are commenting on the need for any form of in-class instruction. The 2017 responses, however, did have in-class instruction in the form of both lectures and team activities; thus, those comments refer to the need for increased time spent in class or the lectures and team activities weren't in depth enough.

The percent of responses complaining about course resources decreased in 2017 compared to 2016, though minimal changes were made to those items (the online textbook was updated to a newer version and the overall number of problems in MPL were reduced). The category "too much material" includes both time-based workload comments, as well as the perceived complexity of the given material.

Discussion

The first quantity being analyzed in this study is whether student learning improved from 2016 to 2017. Analysis of the full homework dataset determined little change in average scores from students between years. However, analysis on the project-exclusive dataset indicates a significant improvement in scores from 2016 to 2017. In either case, the inclusion of team-activities does not appear to have had any negative impact on student learning on either the simpler or more complex homework problems.

The other metric being analyzed in this study is whether student participation on the homework assignments changed between years. Both the entire-homework and project-exclusive analyses found no significant differences between the number of attempted problems, indicating that students didn't participate in the homework any differently given the varying course structure. This is likely due to the same platform, MPL, that was used in both years, which may be the determining factor in how students participated in these assignments. Looking at the survey results, student perceptions of the online tool improved dramatically between years.

The research question of how students changed from 2016 to 2017 in their self-reports of effort, time, and value of programming show that students had a significant increase in positivity about programming in effort and value. The positive or neutral comment, *fair amount*, in amount of effort increased in 2017 and the negative, *too much*, decreased in 2017. This shows that students were responding in a more positive way to programming in 2017 than in 2016. Students' responses of *very valuable* and *somewhat valuable* in their perceived value of programming increased in 2017 from 2016, which is another indicator that students responded to programming in a more positive way in 2017. There was not a significant difference in time from 2017 and 2016, which shows that students spent a similar amount of time between the two years on learning programming and that time spent on programming did not change but students viewed it as a fairer work load and of more value in 2017.

Limitations

A limitation of this study is that two changes were made between the two iterations of the course: both team activities and in-class lectures were added to the course simultaneously. Because of this, the authors were unable to determine which of the two changes had a greater effect on student learning. At a minimum, student perceptions in 2017 show that students enjoyed the use of both additions to the course. Most students both said that the in-class lectures and team activities were enjoyable and helped them learn the material, though it is not clear which of the two has a larger impact from those responses.

Another limitation is that student learning was only assessed through homework assignments; there was no other quantitative assessment or test in this course related to programming. Homework is not the optimal way to measure student learning as students can collaborate and copy from peers, use online resources, etc. Except for programming project scores, student performance (homework) scores and attempts did not change significantly between the two years. The project-exclusive assessment shows a significant improvement between years, meaning that it's possible that the higher-complexity problems are a better form of looking at this metric.

One issue with the two sets of data analyzed in this study (the homework set and survey results) is the large discrepancy in samples. While all the instructors and TAs used the same lecture slides and received similar training, each has varying programming experience and lectures cannot be guaranteed to be given in an identical manner across sections. Also, only half (3 of 6) of the instructors and TAs were common in both 2016 and 2017, which further varies how the material was delivered due to different teaching styles of instructors.

Future Directions

Several improvements could be made to better instruct programming to freshmen engineering students. After student feedback suggested that more in-class learning would be helpful, it was added in 2017, and after that iteration students still felt that there could be more. One potential improvement could be to further increase the amount of programming work done in class, both in terms of team activities and instructor-led lectures. This could also help with the other commonly discussed issue, which was the MPL platform. Students appeared to dislike the online textbook offered through the programming software.

Another distinction that can be made regarding this study is its basis on the concepts of collaborative learning through “pair programming”, which was expanded to group work in teams of 4-5 students. There is a potential that increasing the size of groups to improve student learning saturates before the numbers used in this study. If this is the case, it may be more appropriate to use groups of 2-3 students.

Additional topics of interest include the impact of collaborative activities on students’ learning of programming for students of different genders, racial and ethnic minorities, majors, and varying degrees of academic preparation.

Conclusion

This paper examines the effects of introducing team-based activities to the instruction of programming to freshmen engineering majors. While no significant differences were found in homework scores after adding the in-class team activities, except when examining only the more difficult projects assigned at the end of Chapters’ 2-5 homework sets, students had a generally positive perception of the use of collaborative learning in the course. Student attitudes related to programming improved between years, though this could be due to the introduction of either the team activities and/or lectures. In either case, the improvement in student performance on programming projects combined with improved student perceptions makes for an overall course improvement.

Works Cited

[1] L. Williams and R. Upchurch, "Extreme Programming for Software Engineering Education?," *Education,* pp. 12-17, 2001.

[2] J. T. Nosek, "The case for collaborative programming," *Communications of the ACM,* vol. 41, no. 3, pp. 105-108, 1998.

[3] C. Mcdowell, L. Werner, E. Bullock, Heather and J. Fernald, "The impact of pair programming on student performance, perception and persistence," *Proceedings of the 25th International Conference on Software Engineering,* pp. 602-607, 2003.

[4] J. Nawrocki and A. Wojciechowski, "Experimental evaluation of pair programming," *European Software Control and Metrics Escom,* p. 99–101, 2001.

[5] B. Isong, T. Moemi, N. Dladlu, N. Motlhabane, O. Ifeoma and N. Gasela, "Empirical confirmation of pair programming effectiveness in the teaching of computer programming," *Proceedings - 2016 International Conference on Computational Science and Computational Intelligence, CSCI 2016,* pp. 276-281, 2017.

[6] L. Williams, R. R. Kessler, W. Cunningham and R. Jeffries, "Strengthening the Case for Pair-Programming," *IEEE Software,* Vols. July-Augus, no. August, pp. 19-25, 2000.

[7] D. Sanders, "Student Perceptions of the Suitability of Extreme and Pair Programming," *Extreme Programming Perspectives,* pp. p. 168-174, 2002.

[8] E. A. Chaparro, A. Yuksel, P. Romero and S. Bryant, "Factors Affecting the Perceived Effectiveness of Pair Programming in Higher Education," *17th Workshop of the Psychology of Programming Interest Group,* no. June 2005, pp. 5-18, 2005.

[9] J. C. Schlimmer, J. B. Fletcher and L. A. Hermens, "Team-Oriented Software Practicum," *IEEE Transactions on Education,* vol. 37, no. 2, pp. 212-220, 1994.

[10] L. Thomas, M. Ratcliffe and A. Robertson, "Code warriors and code-a-phobes," *ACM SIGCSE Bulletin,* vol. 35, p. 363, 2003.

[11] L. A. Williams and R. R. Kessler, "Effects of 'pair-pressure' and 'pair-learning' on software engineering education," *Software Engineering Education Conference, Proceedings,* pp. 59-65, 2000.

[12] C. Mcdowell, L. Werner, H. E. Bullock, J. Fernald, R. Etention and C. Onfidence, "Pair Programming Improves Student Retention, Confidence, and Program Quality," vol. 49, no. 8, 2006.

[13] L. L. Werner, B. Hanks and C. McDowell, "Pair-programming helps female computer science students," *Journal on Educational Resources in Computing,* vol. 4, no. 1, pp. 4-es, 2004.

[14] C. McDowell, L. Werner, H. Bullock and J. Fernald, "The effects of pair-programming on performance in an introductory programming course," *ACM SIGCSE Bulletin,* vol. 34, no. 1, p. 38, 2002.

[15] J. C. Carver, L. Henderson, L. He, J. Hodges and D. Reese, "Increased retention of early computer science and software engineering students using pair programming," *Proceedings of the 20th Conference on Software Engineering Education & Training,,* pp. 115-122, 2007.