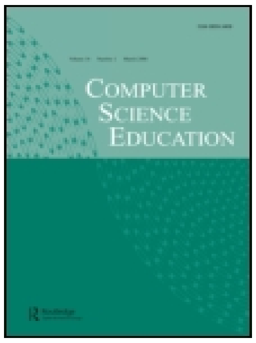


Computer Science Education

ISSN: 0899-3408 (Print) 1744-5175 (Online) Journal homepage: http://www.tandfonline.com/loi/ncse20

Is pair programming more effective than other forms of collaboration for young students?

Colleen M. Lewis

To cite this article: Colleen M. Lewis (2011) Is pair programming more effective than other forms of collaboration for young students?, Computer Science Education, 21:2, 105-134, DOI: 10.1080/08993408.2011.579805

To link to this article: http://dx.doi.org/10.1080/08993408.2011.579805

Published online: 17 Jun 2011.

Computer Science Education
Vol. 21, No. 2, June 2011, 105–134

Is pair programming more effective than other forms of collaboration for young students?

Colleen M. Lewis*

Graduate School of Education, University of California – Berkeley, Berkeley, USA

(*Received 1 November 2010; final version received 1 April 2011*)

This study investigates differences between collaboration methods in two summer enrichment classes for students entering the sixth grade. In one treatment, students used pair programming. In the other treatment, students engaged in frequent collaboration, but worked on their own computer. Students in the two treatments did not differ significantly in their performance on daily quizzes or responses to attitudinal survey questions. However, the students who worked on their own computer completed exercises more quickly than those using pair programming. This study compares two learning environments with high levels of collaboration to isolate aspects of pair programming that are and are not responsible for the reported success of educational research focused on pair programming. This study expands our understanding of pair programming by moving beyond simplistic comparisons of learning environments with and without collaboration and by extending pair programming research to elementary school students.

Keywords: pair programming; collaboration; Scratch

1. Introduction

There is recent enthusiasm within computer science education about adopting and researching the software engineering practice of pair programming. Pair programming is the practice where two programmers work together to solve problems using a single computer. Typically, these programmers will alternate which individual is using the keyboard and mouse, a role referred to as the "driver", and which individual is providing support to identify errors or opportunities for improvement, a role referred to as the "navigator".

Prior research has shown that students using pair programming have increased competence with computer science concepts (Braught, Wahls, & Marlin Eby, 2011), have higher grades (Mendes, Al-Fakhri,

*Email: colleenl@berkeley.edu

ISSN 0899-3408 print/ISSN 1744-5175 online
© 2011 Taylor & Francis
DOI: 10.1080/08993408.2011.579805
http://www.informaworld.com

& Luxton-Reilly, 2006), are more likely to complete the course (Carver, Jenderson, He, Hodges, & Reese, 2007), have increased enjoyment of programming (McDowell, Werner, Bullock, & Fernald, 2003), and have more positive views of their individual performance (Braught et al., 2011).

These benefits are often attributed to the opportunity for increased collaboration (e.g. Preston, 2006). While collaboration has long been shown to be beneficial for students' learning (e.g. Johnson, Johnson, & Smith, 1991), pair programming requires a trade-off of hands-on experience for increased collaboration. This study compares two learning environments with high levels of collaboration to isolate aspects of pair programming that are and are not responsible for the reported success of pair programming in education research.

We examined the differences between pair programming and collaboration without pair programming during a summer enrichment program for students entering the sixth grade. In the first treatment, "pair programming", students worked with a partner each day using a single computer. The students in the pair programming condition switched which student was using the keyboard and mouse every 5 minutes. In the second treatment, "intermittent collaboration", students were assigned a partner; however, they completed all tasks on their own computers. These students were required to discuss any problems with their partner before asking an instructor for help and were required to stop programming once every 5 minutes to discuss any recent progress or challenges with their partner.

The two treatments in the current study were designed so that each has hypothesized benefits over the other. It is possible that the pair programming treatment is superior because it provides an increased requirement of collaboration. It is also possible that the intermittent collaboration treatment is superior because it provides students with more individual experience in programming. This is the first known research that examines the trade-offs between two collaborative educational contexts in computer science. To build upon the body of research showing the benefits of collaboration in other domains (Linn & Hsi, 2000; Vygotsky, 1978), research needs to extend beyond comparing pair programming with conditions with no collaboration.

The article presents data collected from daily quizzes, pacing information collected from the online curriculum, survey responses related to students' attitudes about programming, and observations by the course instructors. Students in the two treatments did not differ significantly in their performance on daily assessments or responses to attitudinal survey questions. However, students in the intermittent collaboration treatment completed exercises more quickly than the students in the pair programming treatment. These findings suggest the existence of alternative methods of collaboration that can support novice computer science students. Developing understanding of the costs and benefits of various collaboration techniques holds promise for improving the quality of computer science education and is an important direction for research.

2. Previous research

2.1. The strengths and weaknesses of each treatment

The current study compares two well-motivated educational environments to develop our understanding of the boundaries and affordance of the pedagogical technique of pair programming. Both treatments include ample opportunities for collaboration that can help students breach challenges (Schoenfeld, 1985), gain experience discussing technical concepts and plans (Chi, de Leeuw, Chiu, & La Vancher, 1994), and utilize support to execute tasks within their zone of proximal development (Vygotsky, 1978). While both treatments in the current study involve and require collaboration, the pair programming condition has greater opportunities for collaboration, and the individual condition has greater opportunities for individual exploration and experience. In the following section, we summarize previous research that characterizes the strengths and weaknesses of the pair programming treatment and the intermittent collaboration treatment.

2.1.1. Pair programming – strengths

While both treatments involve collaboration, pair programming has an increased requirement for collaboration, which may afford additional opportunities for students to:

- support and be supported by a learning partner, which can enable competence on tasks that would not be possible individually (Vygotsky, 1978);
- avoid a wild-goose chase during the problem solving process (Schoenfeld, 1985);
- engage in self- and peer explanations, which is helpful for conceptual development (Chi et al., 1994);
- learn from peer explanations that may be better matched to students' existing understanding (Linn & Hsi, 2000);
- view computer science as a collaborative, rather than isolating, process, which is a common misconception regarding computer science (Margolis & Fisher, 2003);
- describe a programming process in human-understandable terms, a core competency in learning to program (Soloway, 1986), and

- plan changes to code with a partner rather than making random or trivial changes to code when facing a challenge, a pattern frequently observed by educators and in previous research (Jadud, 2006).

2.1.2. Individual work – strengths

While the intermittent collaboration treatment in this study required some collaboration, the requirements were less significant than in the pair programming treatment. The realization of the above benefits of pair programming may be mediated by the decreased time engaged in collaboration. However, with decreased collaboration came increased opportunities for students to:

- exercise greater control over their individual learning, which neurology research has shown is advantageous for learning because it involves recruitment of distinct neural systems (Voss, Gonsalves, Federmeier, Tranel, & Cohen, 2010);
- increase their intrinsic motivation by taking ownership and pride in an accomplishment (Deci, 1971);
- pursue their individual interests within a project, which may increase enjoyment and reinforce learning (diSessa, 2000), and
- progress at a differentiated pace determined by their individual understanding and competence (Tomlinson & Allan, 2000).

In addition to these advantages of individual learning, collaboration can have drawbacks, for example:

- it is difficult to construct a learning environment where pairs engage in productive discourse to make progress in their conceptual understanding (Linn & Hsi, 2000);
- students, particularly children, may lack the skills to engage in meaningful collaborative discourse (Johnson et al., 1991), and
- opportunities for collaboration may inadvertently lead to off-task behavior or conflict (Lemov, 2010).

2.3. Researching a young population

Research has found that students in middle school are at a critical point in shaping their interest in Science, Technology, Engineering, and Mathematics (STEM) fields (Jacobs, 2005). There are three known studies that investigate pair programming at the middle-school level (Denner & Werner, 2007; Werner, Campe, & Denner, 2005; Werner & Denning, 2009). These studies examined the behaviors of pair programming with middle school students and considered learning gains

(Werner et al., 2005) and patterns of interaction and problem solving (Denner & Werner, 2007; Werner & Denning, 2009). They found that students developed competence with some information technology content and found significant evidence of persistence patterns amongst the pairs when facing challenges. These studies extend pair programming research to this population, which is an important target for intervention (Jacobs, 2005) and may present unique constraints when applying research of adults using pair programming.

With a similar population, Inkpen, Booth, Gribble, and Klawe (1995) compared the role switching behaviors of pairs of boys and girls in middle school. These participants were observed sharing a computer with two computer mice to play a puzzle-solving computer game and were placed in one of two sharing conditions. In one condition, the students who were not currently in control could click their mouse button and take control from their partners. In the other condition, the player in control had to click a mouse button to give control to his or her partner. They measured the number of puzzles completed by each pair. They found that pairs of boys were most successful in the game in the condition where they could take control and that pairs of girls were most successful in the condition where they could give control. This study highlights the complexity of collaborative relationships. In contrast to this research, the current study attempts to remove patterns of role switching from consideration by imposing potentially unnatural, but consistent, rules for role switching.

These studies are the only other known studies investigating pair programming or computer sharing with this age population. This is in stark contrast to the prevalence of outreach efforts targeting students at this age. Additional research should mirror research at the college level to investigate the application of pedagogical techniques with this population.

3. Research questions

The following two research questions are parallel in structure with an emphasis on students' learning, attitudes, and pace.

- Do the increased levels of collaboration in the pair programming treatment produce greater learning outcomes, greater interest and confidence in computer science, and an accelerated pace of task completion?
- Do the increased opportunities for individual task execution in the intermittent collaboration treatment produce greater learning outcomes, greater interest and confidence in computer science, and an accelerated pace of task completion?

4. Methods

After introducing the curriculum and treatment groups, we present data sources and analysis methods regarding the three main analytic themes: students' understanding of programming constructs derived from daily quizzes, pacing information derived from students' online curriculum logs, and students' attitudes toward computing derived from surveys. Statistical significance testing is determined using a Mann–Whitney–Wilcoxon test, which is similar to a *t*-test without the assumption that the data are normally distributed. A standard threshold of 5% is used for statistical significance.

4.1. Summer enrichment program

The study took place in two offerings of a computer programming course at a summer enrichment program. The summer enrichment program was designed for academically advanced students, and the current course was restricted to students entering the sixth grade. The courses met 4 days a week over a 3-week period. Each of the 12 class days had 3 hours of class time. The study took place in the morning and afternoon sessions in the summer of 2010. The author co-taught the course with a colleague, and the two instructors had the support of a volunteer teaching assistant most days.

4.2. The curriculum

The curriculum used by both treatments was identical and had been developed over multiple iterations and used by both middle school students and college students (www.colleenmlewis.com/Scratch). During the evolution of the curriculum, it was adapted to an online delivery system, Moodle. This online delivery system presented students with reading, programming tasks, and online quizzes with immediate feedbacks akin to the activities used in lab-centric instruction (Titterton, Lewis, & Clancy, 2010).

The majority of the class time was dedicated to students using the online curriculum. During the 3-hour class, students spent on average 1 hour and 49 minutes using the online curriculum. Other offline activities included the following:

- quizzes;
- surveys;
- lectures;
- activities from CS Unplugged (http://csunplugged.org/);
- other non-programming activities to reinforce or introduce programming topics.

The course used three programming languages: Scratch, Logo, and a variation of the Scratch programming language referred to as Snap, formerly Build Your Own Blocks or BYOB (Harvey & Mönig, 2010). Scratch was the primary language in the course; approximately 3 hours of class time was dedicated to learning Snap, and approximately 30 minutes to experimenting with Logo.

Table 1 shows the outline of the curriculum, describing brief details of the skills covered and the specific programming tasks. The curriculum was designed to cover new content each day and provide an opportunity for open-ended practice at the end of the day. Depending upon students' pace throughout the activities, the open-ended practice activity would vary in duration. This allowed students in the class to progress through the curriculum together, despite differences in pace during a single class session.

Sample quiz questions appear in the Appendix, and sample programming tasks are described below to provide concrete examples of the tasks undertaken for purposes of methodological rigor (Nawrocki & Wojciechowski, 2001) and to highlight themes emphasized in the curriculum for use by educators.

4.2.1. Emphasis on iteration

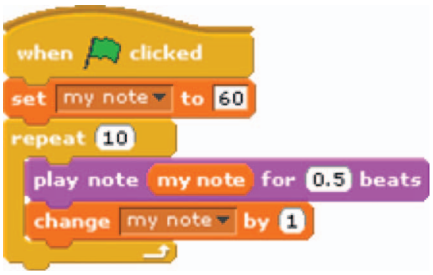
A programming concept that appeared throughout the curriculum was that of iteration. Beginning on the second day of class, students began writing procedures that iterated through a series of values. Table 2 provides a description of multiple iteration tasks and solutions written in Scratch. Iteration was emphasized throughout the curriculum for two

Table 1. Overview of topics covered on each day of the 12-day course.

Day	Topics
1	Introduction to Scratch, sounds, songs, and repeat
2	Variables, iterator pattern, and event-based programming
3	Drawing regular polygons and drawing shapes with iterators
4	Using variables to draw shapes – Part 1
5	Using variables to draw shapes – Part 2
6	Coordinates moving the character with the keys and implementing a game of tag
7	Random, broadcast, making plays, and implementing a rock paper scissors game
8	Animation, lists, and implementing a number guessing games
9	Scratch platform game techniques: flying and jumping and Snap: composition of functions and functions returning numbers or Boolean values
10	Drawing a brick wall and iterator pattern with sentences
11	Final projects – Part 1
12	Final projects – Part 2 and Open House

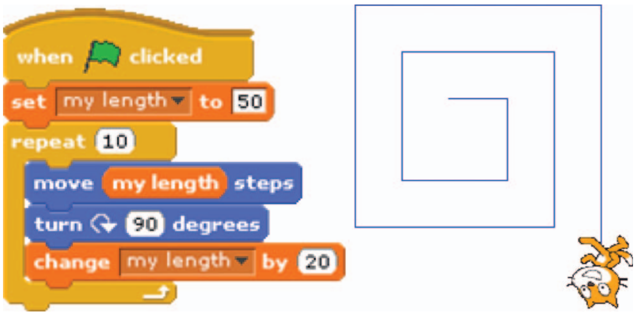
Table 2. Example programs from specific days within the curriculum.

Day 2: Play the notes from 60 to 69.



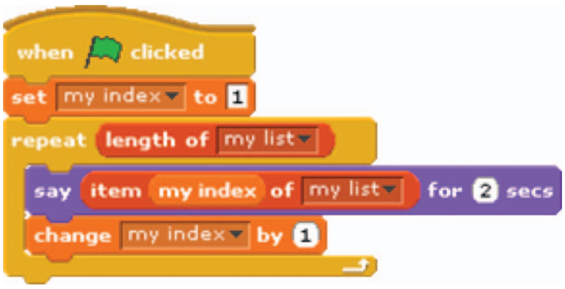
```
when [flag] clicked
set my note to 60
repeat 10
    play note (my note) for 0.5 beats
    change my note by 1
```

Day 3: Draw a squiral (square + spiral).



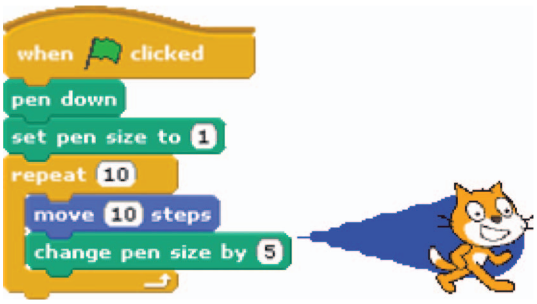
```
when [flag] clicked
set my length to 50
repeat 10
    move (my length) steps
    turn ↻ 90 degrees
    change my length by 20
```

Day 5: Say every item in a list.



```
when [flag] clicked
set my index to 1
repeat (length of my list)
    say (item (my index) of my list) for 2 secs
    change my index by 1
```

Day 8: Draw a line that increases in thickness.



```
when [flag] clicked
pen down
set pen size to 1
repeat 10
    move 10 steps
    change pen size by 5
```

reasons. The first is the applicability to professional programming languages. The second is the opportunity for students to practice the pattern across multiple problem contexts, which, based upon previous research, was assumed to be beneficial for learning (diSessa & Wagner, 2005).

4.2.2. Emphasis on iterative development

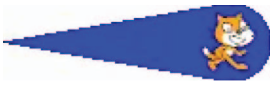
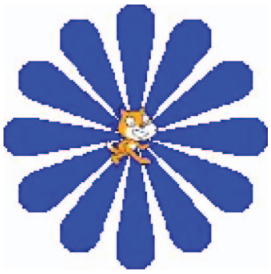
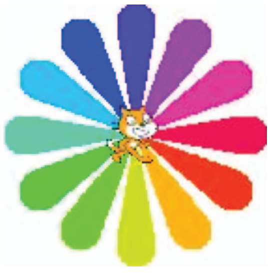
Below we show a series of exercises that students were given on the fifth day of instruction to show the pattern of iterative development, which was emphasized throughout the course. Before beginning the first task, students were shown the goal, an image of a multi-colored flower, shown on the right in Table 3. Each exercise introduced a new level of complexity in accomplishing the final goal. For example, students started out drawing a single petal, and then drew a series of petals, until they finally changed the color of each petal to accomplish the final goal.

This scaffolding of students' work through iterative development was designed to help students be more successful in writing complex programs and to model the process of breaking goals into multiple intermediate tasks.

4.3. Assignment to treatment groups

Before enrolling in the class, parents and students were informed that the instructor would be researching the course. However, they were not aware that there would be differences between the morning and afternoon offerings. Students were, therefore, not able to self-select into a treatment group. This research design is called quasi-experimental (Campbell, Stanley, & Gage, 1966), indicating that assignment to treatment groups is not random, but using existing classroom populations.

Table 3. Iterative development steps in creating the rainbow-colored flower.

Step 1	Step 2	Step 3
		

The final number of class participants was not confirmed until the first day. On the first day of class, the morning section was selected as the pair programming treatment because there was an even number of students. If there had been an odd number of students in the morning class, we planned to have the afternoon class assigned to the pair programming treatment so as to decrease the likelihood that we would have an odd number of students in the pair programming treatment.

Before the class began, two students in each treatment chose not to participate in the research project, leaving 18 students in the pair programming treatment and 22 students in the intermittent collaboration treatment. There were 13 female students in the sample, but differences in gender are not the analytical focus.

4.4. Pair programming treatment

Students in the pair programming treatment worked together on one computer for all programming activities. Students were assigned seats daily. At the beginning of each class day, students sitting at each computer were assigned to be either "Partner A" or "Partner B." Whenever students were programming, a Scratch program was projected that displayed a large "A" or "B" depending upon which partner should be acting as the current driver and using the keyboard and mouse. The Scratch project played music every 5 minutes to indicate that pairs should switch roles. On the first day of class, for six iterations, the Scratch project played music every minute to practice switching driver and navigator. The Scratch project was stopped when announcements were made by the instructors or during the class break so that each partner was ensured 5 minutes as the driver before switching. Five minutes was chosen by the author for the previous offering of the course (Lewis, 2010) as an estimate of what might be reasonable for this population. Based upon the visible impatience of students waiting for their turn as driver in the previous study, we chose to maintain the use of 5-minute intervals. To supplement this experience, it may be more appropriate to base this time on empirical evidence regarding the attention span of the population. In the work of Werner et al. (2005) with middle school girls, 15 minutes was the average duration between switching roles, and they did not report any adverse effects.

Pairs were reassigned each day, and pairing was random except in cases specifically intended to improve consistency of pacing. If a pair of students had difficulty in completing the activities, the next day, the instructors attempted to pair each of those students with a student who had progressed at an accelerated pace. Poor communication, conflict, or off-task behavior often delayed a team's pace, and we estimate that instances of a delayed pace were frequently not the result of a lack of students' understanding.

In the previous iteration of the class, all students used pair programming and were verbally reminded to switch roles every 5 minutes when a timer rang (Lewis, 2010). In that offering, we had difficulty with students losing track of who was the driver and the navigator, using the keyboard and mouse when it was not their turn, and frequently arguing about turn taking. A substantial amount of instructor time was dedicated during that summer to mediating arguments, monitoring role switching, and pairing students to avoid conflict. The instructors observed that these conflicts were not present in the current offering and attributed it to the visual record of who should be using the keyboard and mouse. Even if a student used the keyboard or mouse out of turn, they could easily return to the assigned role by consulting the projected letter.

In the current offering, there remained challenges in the execution of pair programming. Occasionally, we had to remind students to pay attention while their partner was using the keyboard and mouse. A student using the keyboard and mouse would sometimes ask an instructor for help before consulting his or her partner. While this behavior differs from the idea of pair programming, based upon our experience using pair programming with college students, we believe that this lack of attention, when not using the keyboard and mouse, is common.

4.5. *Individual treatment*

Students in the intermittent collaboration treatment worked individually on a computer. At the beginning of each class session, students were assigned a partner. Before a student asked an instructor a question, they were required to have discussed the question with their partner. This was used to encourage students to engage with their partner around debugging tasks.

Similar to the pair programming condition, whenever students were programming, a Scratch project was projected on the screen. This project had the sole purpose of playing music every 5 minutes. When the music played, students were required to stop what they were doing and talk to their partner. On the first day of class, students practiced this, stopping every minute for the first 6 minutes. During this practice, students gave each other a high-five and were encouraged to use the starting line "Hey partner, you stuck on anything?" or "Hey partner, what are you working on?" Most students continued to use one of these lines to initiate conversation throughout the course. Students were expected to discuss their work with their partner the entire time that the music played, which lasted approximately 28 s. When the music began to play, students covered their monitors with the fabric curtains; however, they could lift the curtain on a computer to discuss a particular program.

Few partners made particularly productive use of this time, and some engaged in off-topic discussion. However, between these exchanges, students would frequently discuss and resolve their problems without the help of an instructor, and we hypothesize that the interruptions and opportunities for discussion reinforced the pair relationship. We observed that it was infrequent that a student would consult another student who was not his or her partner, even if he or she was the same distance away as the student's partner.

Partners were assigned each day randomly, except when intended to separate students who engaged in off-task behavior during class. Unlike the pair programming treatment, students' pace was not taken into account when assigning partners.

4.6. Students' understanding

Students took daily quizzes from the second through tenth day of instruction. Quizzes typically lasted 15 minutes, and the amount of time provided on the quizzes was recorded and was standard between the two treatments. If students completed the quiz before the time limit, they would work on worksheets from CS Unplugged, and most students completed quizzes within the time limit. Quizzes were administered at the beginning of class and emphasized content from the previous day. The quizzes had an average of 15.7 questions (min = 11; max = 21), and the majority of the quiz questions asked students to predict the result of a provided Scratch program as is shown in many of the example assessment items in the Appendix.

All quizzes were graded by the author and digitized for statistical analysis. Questions were graded as a single point, with no partial credit. Questions with multiple parts were graded on each part separately. The *mean performance* on each quiz was calculated per treatment, by averaging students' scores in each treatment. A mean performance of 80% would indicate that the average performance within that treatment for a single quiz was 80%. Students' *individual-mean performance* was calculated by determining an individual student's mean percentage correct across all quizzes. An individual-mean performance of 75% would indicate that the student averaged 75% correct across all quizzes. A Mann–Whitney–Wilcoxon test was performed to detect statistical differences between the classes for individual quizzes and for students' individual-mean performance.

4.7. *Students' attitudes*

We hypothesize that positive changes in attitude could be observed in students in one of the treatments, responding more positively to statements regarding their desire to pursue computer science.

On the ninth day, students answered survey questions, shown in Table 4, regarding their confidence in programming and their intentions to pursue learning more computer science. All responses were made on a four-level Likert scale, with the categories of "Agree", "Agree somewhat", "Disagree somewhat", and "Disagree." To simplify the analysis, students' responses were coded from 1 representing "Agree" to 4 representing "Disagree." This coding was necessary to calculate statistics, such as the mean response on each question for the treatment groups. However, this simplification technique may exaggerate the difference between "Agree" and "Agree somewhat" and may underestimate the difference between "Agree somewhat" and "Disagree somewhat".

On the 11th day, students answered additional survey questions, shown in Table 8, regarding their perception of the difficulty of learning particular programming skills and of accomplishing specific programming tasks. All responses were made on the same four-level Likert scale and were coded as aforementioned.

4.8. *Students' pacing*

Logs were collected from the online curriculum and processed in an attempt to identify whether the students in either treatment worked through the curriculum at a faster pace. At 89 points throughout the course, students were prompted to submit a project through the online curriculum. Each of the steps, which prompted students to submit a project, will be referred to as a single *submission point*. A relative duration was calculated for each submission point as the average time between the students' submission for the previous and current submission points. We will consider a student's pace to be determined by the duration between consecutive submission points, despite the fact that there were different activities and different step types between each submission point.

Table 4. Survey questions administered on the ninth day of instruction, using a four-level Likert scale.

I plan to continue to use Scratch after the class.
I am good at writing computer programs.
Writing computer programs is easy.
I want to be a computer scientist.
I want to take another computer programming course.
I understand more about how computer programs and games are made.

If no project was submitted for a submission point, the duration for that and the next submission point was treated as missing data. If a student made a submission out-of-order, any out of order submission point was considered missing data. At the end of each class, students often submitted partial projects to their current submission point. This artificially lowered the mean duration for the submission. As a precaution, we removed any submission from consideration if any students submitted the project within 3 minutes of the end of the time programming that day. Of the 89 submission points throughout the curriculum, we selected 35 that had reasonably complete data. Submission points were rejected if they contained data for less than 70% of the students' submissions. For each of the 35 submissions, we calculated the mean duration for students in each treatment.

We conducted *t*-tests to identify whether the differences in the mean submission time for each treatment were statistically significant at the 5% level. When conducting 35 independent *t*-tests, we would expect that 5% of the 35, approximately 2, would show statistical significance purely based upon random variation. We calculated the standard deviation for the duration of each submission point to determine if there were different patterns of variations within each class.

While we hypothesized that students' pace might vary between the treatment groups, an increased pace does not necessarily represent a core pedagogical goal. For example, if students submitted incomplete or incorrect projects, they may progress at an accelerated place. After exploring opportunities for assessing differences in quality and correctness, we determined that students' submissions were not a viable data source. As aforementioned, the rate of missing submissions was high. In addition, the submissions frequently appeared to be a wrong file or an incomplete version. It is possible that students forgot to save their current file before submission and therefore submitted an incomplete version.

5. Results

The populations in each class were found to be comparable, despite the lack of random assignment. Based upon students' self-report of previous experience, there is no information to suggest that the populations differed in prior programming experience or other relevant characteristics. We further validated comparability by (1) comparing standardized test scores between each treatment, (2) attempting to standardize and measure the amount of time students programmed, and (3) using techniques to ensure that students in each treatment were attentive during announcements and lectures.

We compared anonymous standardized test scores submitted to the program. While the mean for both Mathematics and English tests was slightly higher in the intermittent collaboration treatment, the differences were not significant at the 5% level for either Mathematics ($t(28) = 0.706$, $d = .25$) or English language arts ($t(29) = 1.39$, $d = .25$).

We compared the number of minutes programming across the treatment groups. For most students, in-class programming represented the entirety of their programming experience. No programming homework was assigned and few students reported programming at home. To verify whether students in the treatment groups had similar amounts of in-class programming, logs from the online curriculum were mined to detect the total amount of time students spent programming. On the 11th and 12th days, the students worked on a final project, and there was not sufficient log data from the online curriculum to detect the duration of activities. We found that students in the pair programming treatment spent a total of 17 hours and 53 minutes programming. Students in the intermittent collaboration treatment spent a total of 18 hours and 28 minutes, an average of 3.5 minutes more per day. Given the small difference in the daily averages, the treatments are considered comparable regarding time programming.

We used techniques to standardize students' level of attention. Although the students in the summer program were typically excellent at following instructions, during the previous offering of the course, we had difficulty making announcements while students were in front of their computers. In the current study, the instructors constructed fabric curtains that were used to cover the computer screens. When students began to work, they were told to raise their curtains. When an instructor wished to make an announcement, students were told to lower their curtains. In this way, we eliminated classroom management interruptions during announcements, directions, and lectures and, we believe, ensured that students' level of attention was more consistent.

These three techniques help in validating the comparability of the treatments and the comparability of the populations within each treatment group.

5.1. *Students' understanding*

Figure 1 shows the mean performance within each treatment group for each quiz. Students in the intermittent collaboration treatment had a higher mean performance on eight of the nine quizzes. However, the differences in quiz performance between the treatments were not statistically significant at the 5% level for individual quizzes or students' individual-mean performance.

While the mean of the students' individual-mean performance differs minimally between the pair and the intermittent collaboration treatments, 66.2% and 72.4%, respectively, the differences in distribution are noteworthy. There was a larger range of individual-mean performance among students in the pair programming treatment. Table 5 shows the range of each quartile, and Figure 2 plots each of these quartiles in a standard box plot. Students' mean performance in the pair programming treatment spans from 32.0% to 89.0%, while in the intermittent collaboration treatment, the means span a range of only 53.0–85.2%.

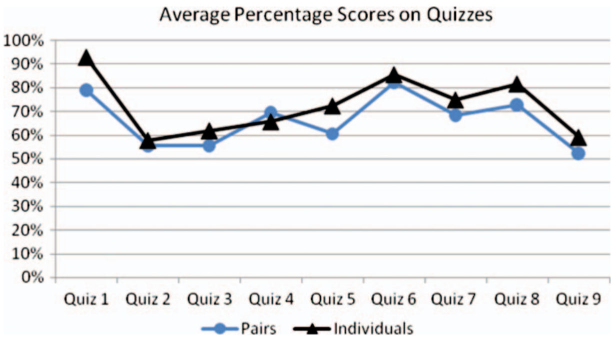


Figure 1. Average percentage score per quiz for each treatment.

Table 5. Distribution of individual students' average quiz percentage.

	Pairs (%)	Individuals (%)
Maximum	89.0	85.2
Q1	75.3	81.1
Median	66.7	74.2
Q3	56.3	66.9
Minimum	32.0	53.0

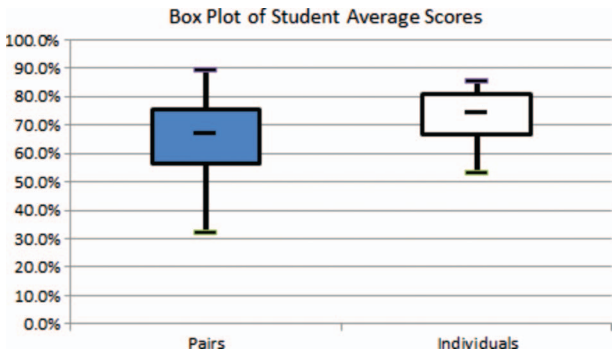


Figure 2. Box plot of individual students' average quiz percentage.

5.2. *Students' attitudes*

5.2.1. *Plans to pursue computer science*

For the three survey questions related to a desire to pursue computer science, the students in the pair programming treatment responded less positively; however, the difference between the treatment groups was not statistically significant.

For each treatment, one student was absent during the survey, so the response rates for the pair and intermittent collaboration treatments were 94% and 95%, respectively. Table 6 shows the average coded response and results of each Mann–Whitney–Wilcoxon test.

Regarding the statement "I plan to continue to use Scratch after the class", 81% of the students from the intermittent collaboration treatment and only 59% of the students from the pair programming treatment responded with "Agree", as shown in Figure 3.

Table 6. Average response to questions regarding goals to continue programming where 1 indicates a response of "Agree" and 4 indicates a response of "Disagree" and results of significance testing using a Mann–Whitney–Wilcoxon test.

	Mean response (1 = "agree"; 4 = "disagree")		Mann–Whitney–Wilcoxon test	
Questions	Pairs	Individuals	*z*	*p*-value
I plan to continue to use Scratch after the class	1.5	1.18	−1.265	0.2059
I want to be a computer scientist	2.39	2.05	−1.182	0.2372
I want to take another computer programming course	1.83	1.45	−1.506	0.1321

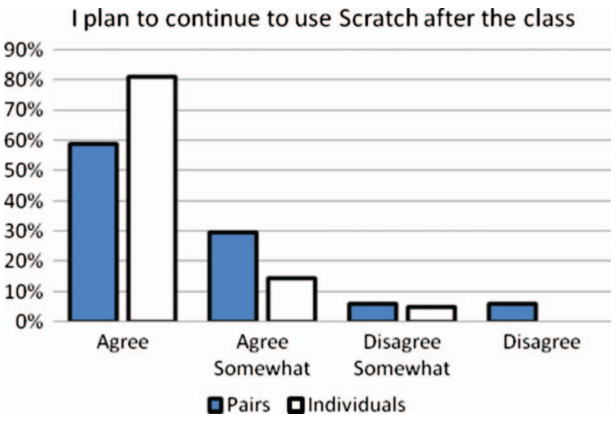


Figure 3. Distribution of responses to the question "I plan to continue to use Scratch after the class".

A similar pattern, shown in Figure 4, occurred regarding the statement "I want to take another computer programming course." Sixty-two percent of students from the intermittent collaboration treatment and only 29% from the pair programming treatment responded with "Agree".

As shown in Figure 5, less than 20% in each class responded with "Agree" regarding the statement "I want to become a computer scientist." However, the distribution of responses for this question showed again that students in the intermittent collaboration treatment are more positive about pursuing computer science. Seventy-six percent of the students in the intermittent collaboration treatment and 47% of the students in the pair programming treatment responded that they "Agree" or "Agree somewhat" with the statement: "I want to be a computer scientist".

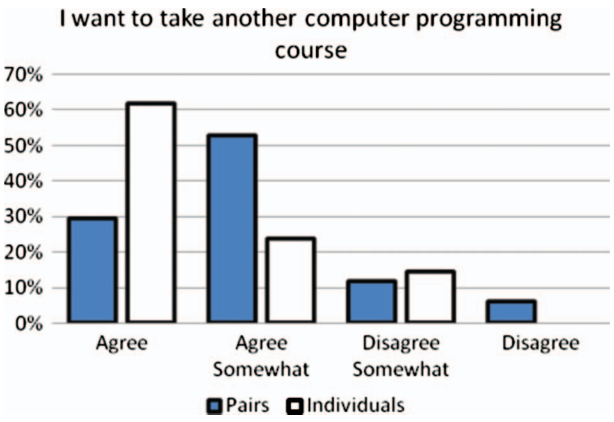


Figure 4. Distribution of responses to the question "I want to take another computer programming course".

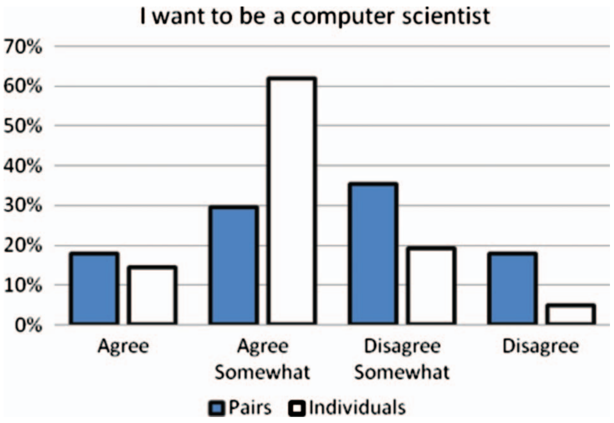


Figure 5. Distribution of responses to the question "I want to be a computer scientist".

5.2.2. *Confidence in programming ability*

The students in the intermittent collaboration treatment were slightly more positive regarding confidence in computer programming, as shown in Table 7. None of the differences were statistically significant. The shape of the distribution of these responses was nearly identical between the two groups, and hence, the graphs have been omitted.

5.2.3. *Perception of difficulty*

On average, students in the pair programming treatment rated the difficulty of various topics as slightly less difficult; however, the differences between these averages were not statistically significant using a Mann–Whitney–Wilcoxon test ($z = 0.545$; $p = 0.586$). The average difficulty ratings for the pair and intermittent collaboration treatments were 2.59 and 2.50, respectively, indicating that students in the pair programming treatment rated survey items as slightly less difficult. The mean coded value for each question was computed for each group and is shown in Table 8, with the larger group mean shown in bold.

A Mann–Whitney–Wilcoxon test was performed on each of the 13 survey questions related to perception of difficulty. Only one question had differences that were significant at the 5% level; however, given the large number of tests, this is assumed to be the result of random variation and not indicative of differences in the perception of the difficulty between the treatment groups.

5.3. ***Students' pacing***

We compared the mean from each treatment and found that for 24 of the 35 submissions or 69%, students in the pair programming treatment had

Table 7. Average response to questions regarding confidence in programming ability, where 1 indicates a response of "Agree" and 4 indicates a response of "Disagree", and results of significance testing using a Mann–Whitney–Wilcoxon test.

	Mean response (1 = "agree"; 4 = "disagree")		Mann–Whitney–Wilcoxon test	
Questions	Pairs	Individuals	z	p-value
I am good at writing computer programs	1.89	2.09	0.770	0.4415
Writing computer programs is easy	2.33	2.55	0.676	0.4990
I understand more about how computer programs and games are made	1.11	1.23	0.279	0.7800

Table 8. Survey questions administered on the 11th day of instruction, using a four-level Likert scale and mean rating, where 1 indicates a response of "Agree" and 4 indicates a response of "Disagree", bold indicates a higher mean response.

	Pair	Individual
It was difficult to learn to use Scratch	2.44	**2.59**
It was difficult to learn to use BYOB	**2.78**	2.41
It was difficult to make the squiral [square + spiral]	**2.56**	1.77
It was difficult to make the rainbow	1.33	**1.68**
It was difficult to learn to use lists	2.56	**2.73**
It was difficult to learn to use *and*	**3.00**	2.82
It was difficult to learn to use *if*	**3.50**	3.27
It was difficult to learn to use variables	**2.78**	2.32
It was difficult to learn to use broadcast	1.17	**2.91**
It was difficult to make the brick wall	**1.78**	1.77
It was difficult to learn binary	2.39	**2.55**
It was difficult to make the flower petals	2.06	**2.68**
It was difficult to draw shapes	**3.39**	3.05
Average rating	**2.59**	2.50

a higher mean time to completion. We calculated the sum of the mean times to complete each submission. The total duration for the 35 submission points for the pair programming treatment was 7 hours and 35 minutes. The total duration for the intermittent collaboration treatment was 6 hours and 33 minutes. That is 13.7% less time is spent by students in the intermittent collaboration treatment across the 35 submission points.

Nine of the 35 *t*-tests indicated the difference between the groups to be statistically significant at the 5% level. Eight of the nine were in cases where the mean duration for the pair programming treatment was higher. We would expect two of these tests to show statistical significance due only to random variation. This suggests that the faster pace of students in the intermittent collaboration treatment was not an anomaly attributable to only random variation. For three of these eight, the result was significant at the 1% level.

For 14 of the 35 submissions or 40%, students in the pair programming treatment had a higher standard deviation. While we found higher variation on assessment performance in the pair programming treatment, there appears no significant differences of variation in pace between the two treatments.

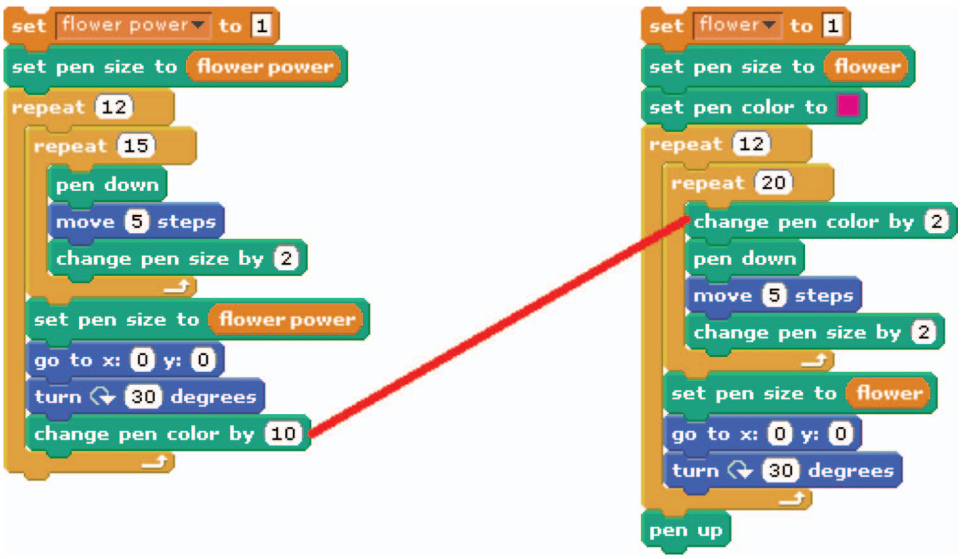
The instructors observed that when students in the intermittent collaboration treatment would get stuck, their partner would show them the correct answer. The students, who were stuck, were then able to copy the answers and move on in the curriculum. For example, for step 3 of the multi-color flower exercise shown in Table 3, two students from the intermittent collaboration treatment who were working as partners submitted identical code. These identical solutions made the mistake of changing the pen color within a single petal rather than between each petal. Given the diversity of other submissions, this would be nearly impossible to occur if the students had not been extensively collaborating.

On the same problem, the two students sitting in the row with these students also showed clear signs of collaboration in their solutions, shown in Figure 6. There are minor differences between their submissions, and no other students in the intermittent collaboration treatment used a variable in this program. The only significant difference between these two submissions is the placement of the "change pen color by" block. The solution on the left correctly places this block to change colors between each petal. The submission on the right makes the same mistake as the two identical submissions described above, by changing the color within each petal. The submission on the right may have been influenced by his or her neighbors, the pair of students previously mentioned, who had made the same mistake. While this was a common mistake, Figure 6 shows clear evidence of collaboration between this partnership and plausible evidence of collaboration between members of the two partnerships.

6. Discussion

6.1. Students' understanding

Previous research has shown greater learning gains for students using pair programming than students working individually (Braught et al., 2011).



```
set flower power to 1
set pen size to flower power
repeat 12
    repeat 15
        pen down
        move 5 steps
        change pen size by 2
    set pen size to flower power
    go to x: 0 y: 0
    turn ↻ 30 degrees
    change pen color by 10
```

```
set flower to 1
set pen size to flower
set pen color to ■
repeat 12
    repeat 20
        change pen color by 2
        pen down
        move 5 steps
        change pen size by 2
    set pen size to flower
    go to x: 0 y: 0
    turn ↻ 30 degrees
pen up
```

Figure 6. Submissions from a pair of students for step 3 in the multicolor flower exercise shown in Table 3.

This pattern did not hold in the current study, where students working on their own computer had support and intermittent collaboration.

While the differences in performance on quizzes between the treatments were not statistically significant, the lowest scoring student in the pair programming treatment scored much lower on average than the lowest scoring students in the intermittent collaboration treatment, averaging 32.0% and 66.2%, respectively. The expectation that pair programming can improve the performance of the weakest students through peer scaffolding does not appear to be enough to explain the results in the current study. This observation of much lower performance by the lowest scoring student in the pair programming treatment violates our expectation about this potential benefit of pair programming.

It is possible that a lack of engagement from the student not using the keyboard and mouse depressed his or her performance in the pair programming treatment, resulting in slightly lower quiz scores and greater variation in quiz performance. While these are most likely attributable to random variation, it is possible that pair programming has a greater differential benefit for students than the intermittent collaboration treatment. In our experience with college students, less experienced students frequently report passively watching their more experienced partner program. These college students are able to articulate their lack of understanding and explain that they rarely ask questions when they do not understand the actions of their partner because they do not want to inconvenience their more experienced partner. While a common narrative amongst our college students using pair programming, this consideration is rarely articulated in the pair programming literature. We hypothesize that this less productive experience is also relevant to the younger population in the study and may explain the result of Braught, MacCormick, and Wahls (2010), who found that students in the lowest quartile of initial competence were most successful on examinations when paired with another student from the lowest quartile. Perhaps the less experienced student in an unbalanced pair will be more likely to sit back and watch his or her partner without understanding, creating a differential benefit of pair programming. Based upon this research, our strategy of dividing up students who were progressing at a slower pace may have increased the difficulty for students who were not progressing as quickly as their peers.

6.2. Differences in attitudes

6.2.1. Plans to pursue computer science

In this study, we observed that students who engaged in pair programming were slightly less positive when asked about pursuing computer science. While this result was not statistically significant, it is of interest because pair programming has been shown to increase enjoyment and interest in computer science (McDowell et al., 2003). It is possible that the hypothesis of increased interest and enjoyment does not hold because it is rare to compare pair programming with another highly collaborative environment or because the current focus on sixth grade students makes previous findings inapplicable.

Students in the pair programming treatment may have been less positive regarding pursuing computer science because of conflicts or discomfort in working with a partner, a decreased sense of ownership for the products of programming, or frustration given the slower pace of task completion. There is some evidence to suggest that students in the intermittent collaboration treatment had a greater sense of ownership. For example, on the second day of instruction, they persistently requested to share their electronic keyboards with the class.

6.2.2. Relationship to prior work

The previous year, the author and co-instructor taught a course with the same name that included programming in Logo and Scratch (Lewis, 2010). In this course, all students pair programmed and shared a computer. The study considered the effects of programming environment on students' understanding, perception of difficulty, and confidence.

One treatment group programmed for 18 hours in Logo, while the other programmed for 18 hours in Scratch. These two programming environment interfaces differ substantially. In Logo, students type commands, while in Scratch, students drag and drop commands to create programs. Despite these differences in user interface, much of the underlying structure of each language is the same, and students in each treatment completed the same programming tasks.

The study found that students who learned Logo were comparatively less competent at answering questions related to conditionals. Despite the Scratch students' better performance, the Logo students were more confident in their programming ability.

Students in the previous study responded to the same prompt as students in the current study: "I am good at writing computer programs." Of the students who learned Scratch in the previous study, 38.5% responded with "Agree", compared to 70.8% of the students who learned Logo and responded with "Agree" for this statement. This finding unearthed the question of whether Scratch, which may be perceived as a game rather than a programming environment, can support students in developing confidence in their programming ability.

Despite the fact that students completed exercises that they knew were used in an introductory course at University of California, Berkeley, in the current study, we see an even smaller percentage of students responded with "Agree" for this statement. Across both treatment groups, only 13.2% of students responded with "Agree" for the statement "I am good at writing computer programs." The distribution of responses did not vary significantly between the treatment groups. These data replicate the pattern of confidence by students learning Scratch in previous work (Lewis, 2010). Future work should investigate children's perception of Scratch and computer programming to evaluate the potential for Scratch to build students' confidence in computer programming.

6.3. Differences in pace

One observable difference between the treatment groups was in differences in pace between points at which students submitted projects through the online curriculum. Students in the pair programming condition operated at a relatively slower pace than students in the individual condition while completing required activities.

One reason for this result may be the overhead of communication in the pair programming treatment. Research has found that students are able to solve more difficult problems when working in pairs (Hanks, 2008); however, negotiations, discussions, and explanations may slow the overall progress of students using pair programming.

However, our most robust hypothesis is that students can more quickly reach a correct solution when they can each experiment on their own computer in parallel. In the Scratch programming language, it is very easy to test whether a current program is correct and to make modifications to retest the program. It is likely that two students working in parallel using this type and trial-and-error technique may stumble upon the solution more quickly than if those two students were working together on a single computer.

While the instructors assumed that this practice of sharing answers would be detrimental for learning, there were no measurable differences between the two treatment groups. This informal practice may account for the accelerated pace of students in the intermittent collaboration treatment.

6.4. Threats to validity

The primary threat to validity is the concern that the population differed between the two treatment groups. There were only 40 students in the study. While there were no statistically significant differences in standardized test scores reported, standardized test scores have been shown to be a poor predictor of success in computer science classes (Simon et al., 2006), and not all students submitted scores. This threat to validity is similar to previous studies that lack information regarding the comparability of initial populations (e.g. Kuppuswami & Vivekanandan, 2004; Mendes et al., 2006).

While there were no differences detected for students' understanding in this study, as in any study, it is possible that assessments were misaligned with the competencies developed in the treatment groups. For example, no assessment determined students' proficiency in communicating and clarifying plans to a learning partner. This proficiency may have increased to a greater extent for students in the pair programming treatment, but was not measured in this study.

The current study considered the pace of students' submissions without taking into account the quality of those submissions. Based upon a sampling of submission points, this was not utilized as a method within the study because there were a high number of missing submissions and many errors could not reliably be attributed to students' lack of understanding the programming content. For example, occasionally, students appeared to submit the wrong file or an incomplete file. It is not possible to tell in these cases whether the student was unable to complete the activity or whether he or she submitted a file that did not contain the most recent version of his or her work. It may be possible to remove these barriers in future studies by targeting relevant computer literacy skills explicitly.

7. Conclusion

This study compared two highly collaborative learning environments to isolate aspects of pair programming that are influential for supporting students' understanding, attitudes and interest in computer science, and pace. There were no statistically significant differences between students' performance on daily quizzes; however, there was greater variation in performance on daily quizzes for students using pair programming. There were no statistically significant differences between students' responses to attitudinal survey questions; however, students who used pair programming were slightly less positive for survey questions related to interest in computer science. Students who worked on their own computers completed exercises more quickly, but it was not feasible to validate if the quality of students' work was similar between the two treatments.

There are obvious benefits of pair programming over environments without support and collaboration. Alternative forms of collaboration, such as the intermittent collaboration treatment described here, may provide opportunities for improved performance over solitary efforts, without some of the drawbacks of pair programming, such as increased likelihood of conflict and frustration (Lemov, 2010).

The level of conflict observed in the previous offering of the course is a testament to drawbacks of pair programming, and the current study may achieve a more ideal instantiation of pair programming than should be expected in a typical classroom. While some frustration may be germane to learning to collaborate, the levels of conflict in the previous offering seemed to exceed the levels that were pedagogically valuable.

The current study nearly eliminated the need for teacher-mediated conflicts between pairs. While the role-switching techniques used in the current study drastically reduced conflicts in the classroom, the strategy of rigid and public role switching may not be appropriate for older students, and it is likely that any techniques would need to be refined for a given population. Future work should investigate adapting this technique to work with older populations.

The author hypothesize that some instantiations of pair programming are unproductive for students' learning, engagement, and success and that these dysfunctional pairings are common in pair programming relationships. Additional research is required to understand the differential effects of pair programming for students.

Acknowledgments

The author would like to thank her co-instructor Brittany Murlas, their laboratory assistant Navin Eluthesen, and the students and staff of the enrichment program. Andrea diSessa, Michael Clancy, Angie Little, Nathaniel Titterton, Katherine Lewis, and the anonymous reviewers provided substantial reviews of the work, which greatly improved the quality and clarity of the work.

References

Braught, G., MacCormick, J., & Wahls, T. (2010). The benefits of pairing by ability. *ACM Bulletin, SIGCSE 2010*, 249–253.

Braught, G., Wahls, T., & Marlin Eby, L. (2011). The case for pair programming in the computer science classroom. *ACM Transactions on Computing Education*, *11*, 1–21.

Campbell, D.T., Stanley, J.C., & Gage, N.L. (1966). *Experimental and quasi-experimental designs for research*. Chicago: R. McNally.

Carver, J., Jenderson, L., He, L., Hodges, J., & Reese, D. (2007). Increased retention of early computer science and software engineering students using pair programming. In *Proceedings of the 41st technical symposium on software engineering education and training (CSEET'07)* (pp. 115–122). Washington, DC: IEEE Computer Society.

Chi, M.T.H., de Leeuw, N., Chiu, M.H., & La Vancher, C. (1994). Eliciting self-explanations improves understanding. *Cognitive Science*, *18*, 439–477.

Deci, E.L. (1971). Effects of externally mediated rewards on intrinsic motivation. *Journal of Personality and Social Psychology*, *18*, 105–115.

Denner, J., & Werner, L. (2007). Computer programming in middle school: How pairs respond to challenges. *Journal of Educational Computing Research*, *37*, 131–150.

diSessa, A. (2000). *Changing minds: Computers, learning, and literacy*. Cambridge, MA: MIT Press.

diSessa, A.A., & Wagner, J.F. (2005). What coordination has to say about transfer. In J. Mestre (Ed.), *Transfer of learning from a modern multi-disciplinary perspective* (pp. 121–154). Greenwich, CT: Information Age Publishing.

Hanks, B. (2008). Problems encountered by novice pair programmers. *Journal of Educational Resources in Computing*, *7*, 1–13.

Harvey, B., & Mönig, J. (2010). Bringing 'No Ceiling' to Scratch: Can one language serve kids and computer scientists? *Constructionism, 2010*, 1–10.

Inkpen, K., Booth, K.S., Gribble, S.D., & Klawe, M. (1995). Give and take: Children collaborating on one computer. In *ACM CHI* (pp. 258–259). New York, NY: ACM.

Jacobs, J.E. (2005). Twenty-five years of research on gender and ethnic differences in math and science career choices: What have we learned? *New Directions for Child and Adolescent Development*, *2005*, 85–94.

Jadud, M.C. (2006). Methods and tools for exploring novice compilation behaviour. In *International Computing Education Research Workshop, ACM* (pp. 73–84). New York, NY: ACM.

Johnson, D.W., Johnson, R.T., & Smith, K.A. (1991). *Cooperative learning: Increasing college faculty instructional productivity* (ASHE-ERIC Higher Education Rep. No. 4). Washington, DC: The George Washington University.

Kuppuswami, S., & Vivekanandan, K. (2004). The effects of pair programming on learning efficiency in short programming assignments. *Informatics in Education*, *3*, 251–266.

Lemov, D. (2010). *Teach like a champion: 49 techniques that put students on the path to college*. San Francisco, CA: Jossey-Bass.

Lewis, C.M. (2010). How programming environment shapes perception, learning and goals: Logo vs. Scratch. *ACM SIGCSE Publication*, *41*, 346–350.

Linn, M.C., & Hsi, S. (2000). *Computers, teachers, peers: Science learning partners*. Mahwah, NJ: Lawrence Erlbaum Associates.

Margolis, J., & Fisher, A. (2003). *Unlocking the clubhouse: Women in computing*. Cambridge, MA: The MIT Press.

McDowell, C., Werner, L., Bullock, H., & Fernald, J. (2003). The impact of pair programming on student performance, perception and persistence. In *Proceedings of the 25th International Conference on Software Engineering* (pp. 602–607). Washington, DC: IEEE Computer Society.

Mendes, E., Al-Fakhri, L., & Luxton-Reilly. (2006). A replicated experiment of pair-programming in a second-year software development and design computer course. *Proceedings of the Special Interest Group in Computer Science Education*, *38*, 108–112.

Nawrocki, J., & Wojciechowski, A. (2001). Experimental evaluation of pair programming. Paper presented at the 12th European Software Control and Metrics Conference, London, UK.

Preston, D. (2006). Using collaborative learning research to enhance pair programming pedagogy. *ACM SIGITE Newsletter*, *3*, 16–21.

Schoenfeld, A.H. (1985). *Mathematical problem solving*. Orlando, FL: Academic Press.

Simon, Cutts, Q., Fincher, S., Haden, P., Robins, A., Sutton, K., Baker, B., . . . Tutty, J. (2006). The ability to articulate strategy as a predictor of programming skill. In *Proceedings of the 8th Australian computing education conference*. Darlinghurst, Australia: Australian Computer Society, Inc.

Soloway, E. (1986). Learning to program = learning to construct mechanisms and explanations. *Communications of the ACM*, *29*, 850–858.

Titterton, N., Lewis, C.M., & Clancy, M. (2010). Benefits of lab-centric instruction. *Computer Science Education*, *20*, 79–102.

Tomlinson, C.A., & Allan, S.D. (2000). *Leadership for differentiating schools and classrooms*. Alexandria, VA: Association for Supervision & Curriculum Development.

Voss, J.L., Gonsalves, B.D., Federmeier, K.D., Tranel, D., & Cohen, N.J. (2010). Hippocampal brain-network coordination during volitional exploratory behavior enhances learning. *Nature Neuroscience*, *14*, 115–120.

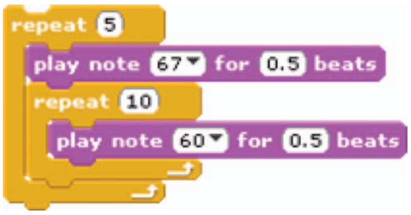
Vygotsky, L.S. (1978). *Mind in society: The development of higher psychological processes*. Cambridge, MA: Harvard University Press.

Werner, L., & Denning, J. (2009). Pair programming in middle school: What does it look like? *Journal of Research on Technology in Education*, *42*, 29–49.

Werner, L.L., Campe, S., & Denner, J. (2005). Middle school girls + games programming = Information Technology fluency. In *SIGITE* (pp. 301–305). New York, NY: ACM.

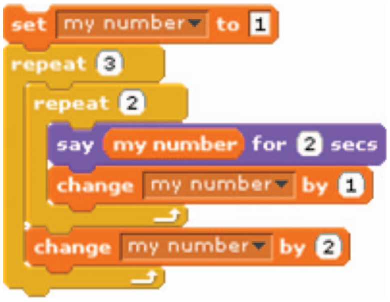
Appendix. Sample questions from each of the nine quizzes

Quiz 1: How many times does the note 67 play when you double click the script below?



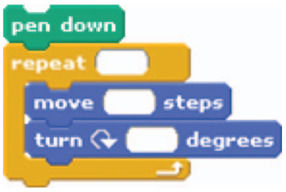
```
repeat 5
    play note 67 for 0.5 beats
    repeat 10
        play note 60 for 0.5 beats
```

Quiz 2: What numbers does the character say when you double click the script below?



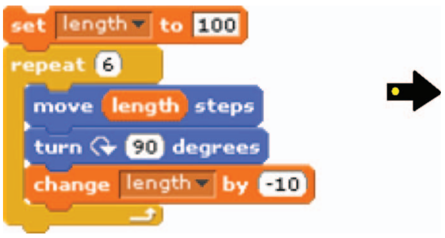
```
set my number to 1
repeat 3
    repeat 2
        say my number for 2 secs
        change my number by 1
    change my number by 2
```

Quiz 3: Fill in the blanks below to draw a 10-sided shape.



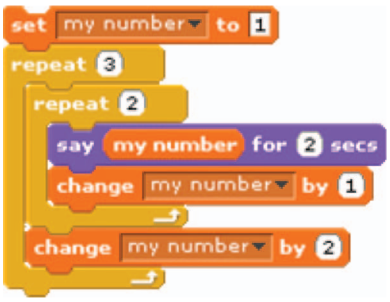
```
pen down
repeat ( )
    move ( ) steps
    turn ↻ ( ) degrees
```

Quiz 4: The character starts at the arrow, facing the same direction as the arrow. Draw what the character draws when you double click the script.



```
set length to 100
repeat 6
    move length steps
    turn ↻ 90 degrees
    change length by -10
```

Quiz 5: What numbers does the character say when you double click the script below?

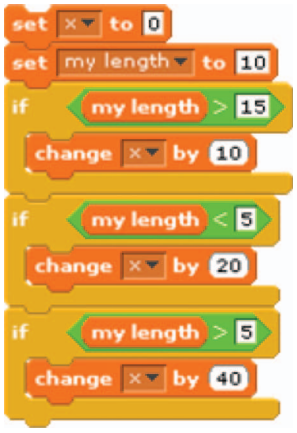


```
set my number to 1
repeat 3
    repeat 2
        say my number for 2 secs
        change my number by 1
    change my number by 2
```

(*continued*)

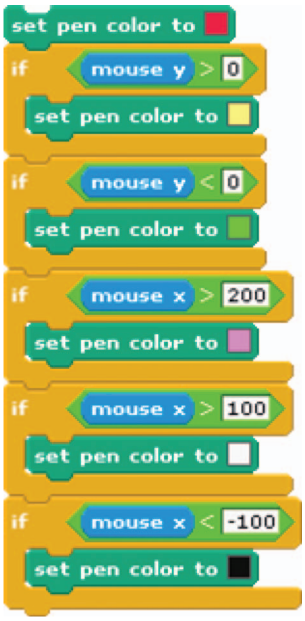
Appendix. (*Continued*).

Quiz 6: What is the value of x after double clicking on the script below?



```
set x to 0
set my length to 10
if my length > 15
    change x by 10
if my length < 5
    change x by 20
if my length > 5
    change x by 40
```

Quiz 7: If the script below is double clicked, what color would the pen be if the mouse is at each position described in the table below? [table omitted]



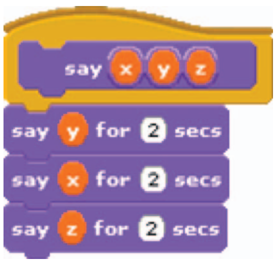
```
set pen color to 🟥
if mouse y > 0
    set pen color to 🟨
if mouse y < 0
    set pen color to 🟩
if mouse x > 200
    set pen color to 🟪
if mouse x > 100
    set pen color to ⬜
if mouse x < -100
    set pen color to ⬛
```

Quiz 8: Draw a script that would say every element in the player-list, no matter how long the player-list is. You can use the variable "index".

(*continued*)

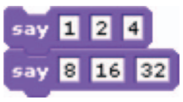
Appendix. (*Continued*).

Quiz 9: We have made this new block:



```
say (x) (y) (z)
say (y) for 2 secs
say (x) for 2 secs
say (z) for 2 secs
```

What will it say when we double click on the script below?



```
say 1 2 4
say 8 16 32
```