The Interface Design of a Collaborative Computer Science Learning Environment for Elementary Aged Students

Amanda Bradbury[1], Eric Wiebe[1], Jessica Vandenberg[1], Jennifer Tsan[1], Collin Lynch[1], Kristy Boyer[2]
North Carolina State University[1] and University of Florida[2].

There is a currently a shortage of computer science professionals and this shortage is projected to continue into the foreseeable future as not enough students are selecting computer science majors. Researchers and policy-makers agree that development of this career pipeline starts in elementary school. Our study examined which collaborative programming setup, pair programming (two students collaborate on one computer) or side-by-side programming (two students collaborate on the same program from two computers), fifth-grade students preferred. We also sought to understand why students preferred one method over the other and explored ideas on how to effectively design a collaborative programming environment for this age group. Our study had participants first engage in five instructional days, alternating between pair and side-by-side programming, and then conducted focus groups. We found that students overwhelmingly preferred side-by-side programming. We explain this using self-determination theory which states that behavior is motivated by three psychological needs: autonomy, competence, and psychological relatedness which side-by-side programming was better able to meet.

INTRODUCTION

Computer science and information technology occupations are expected to grow by 13% between 2016 and 2026; however, current estimates suggest we may not have enough skilled workers to fill this growing demand, as such, more computer science graduates are needed (Baser, 2013; Bureau of Labor Statistics, 2018). One reason students may not select a computer science-related major could be the perception of computer science as difficult, boring, and unsocial (Baser, 2013). Fortunately, past research has shown that these negative perceptions can be combatted through early exposure to computer science concepts in K-12 education (Ashcraft, 2012; Google Inc. & Gallup Inc., 2015).

However, there is a lack of formal computer science educational opportunities in U.S. schools (Ashcraft, 2012). For instance, only 52% percent of 7th to 12th grade students reported computer science classes at their school (Google Inc. & Gallup Inc., 2015). The lack of early exposure to computer science is likely causing many prospective computer scientists to never fully consider the profession, thereby limiting the additions of skilled computer science professionals. Past research with upper elementary school students has demonstrated that this age group is capable of both learning and applying computer science concepts (Grover & Pea, 2013). For instance, Wilson and colleagues (2012) found that students were able to learn computer science concepts such as loops, conditionals, and user interaction, from creating a game in the visual block-based programming environment Scratch. Visual block-based programming allows students to learn complex computer science concepts without having to also learn a complex written programming language whose syntax (spelling, comma placement, indentations, spaces, etc.) is crucial for the program to function correctly (Hill et al., 2015). Although there is clear evidence regarding young students' ability to learn and conceptualize computer science, there is substantially less research exploring elementary aged students' ability to effectively collaborate during programming activities (K-12 CS Framework, 2016).

Modern computer science is a collaborative discipline, with programmers periodically working on the same code simultaneously; therefore, the need for computer scientists to possess collaboration skills in addition to programming skills is critical and increasingly emphasized in the computer science industry (Dingsøyr & Dybå, 2012). Thus, the importance of collaboration is highlighted in the current K-12 Computer Science Framework (2016). For instance, past studies have shown that when students engage in collaborative programming activities, they complete assignments faster, report less effort in completing those assignments (Nawrocki et al., 2005), describe an overall better learning experience (Williams et al., 2002), and demonstrate increased retention of the material (McDowell et al., 2006).

Past Research

Collaborative programming can take the form of many paradigms; two types are pair programming and side-by-side programming. Pair programming is defined as two students working on a program from one computer with one student acting as the driver and the other acting as the navigator (Williams et al., 2002.) The driver has control of the mouse and keyboard and actively constructs and changes the code while the navigator observes the driver and actively seeks to identify errors, plans ahead, and offers suggestions. The driver and navigator should talk through each problem the entire time. Side-by-side programming offers an alternative collaborative configuration. It is defined as two students sitting next to each other, each with their own computer, working on the same program simultaneously (Nawrocki et al., 2005). There is currently very little research looking at collaborative programming as a teaching paradigm for elementary aged students.

Most collaborative programming studies have taken place in either industry or university settings, very rarely including elementary aged learners (Salleh et al., 2011). Research with university students indicated that pair programming led to increased course completion, pass rates, persistence, confidence, and enjoyment of programming (McDowell et al., 2006). Among high school students, pair programming led to increased understanding of computer science concepts and improved students' attitudes towards

programming compared to solo programming (Papadakis, 2018). Research with middle school students found that pair programming significantly increased computational thinking skills and programming knowledge; this effect was especially true for less experienced students (Denner et al., 2014). Lastly, research with elementary aged students found that this age group had difficulty collaborating effectively while pair programming (Tsan et al., 2018). For instance, they often failed to transition effectively between the driver and navigator roles, take or relinquish control at appropriate times, and often did not ask relevant questions (Tsan et al., 2018). Pair programming appears to be an effective teaching paradigm for middle school and beyond; however, similar findings are not present for elementary aged students, possibly due to the shortage of research within this age group. However, it is also possible that pair programming is not effective for elementary aged students because they are not as developmentally advanced as older children and adults; and therefore, do not yet possess the skill necessary for effective collaboration.

Side-by-side programming has been studied significantly less than pair programming, and the research that has been completed mostly involved university students or programmers already in industry. For instance, a study of senior undergraduate computer science students found that they were able to complete assignments faster and reported less effort with side-by-side programming compared to pair and solo programming (Nawrocki et al., 2005). To our knowledge no studies have examined side-by-side programming with elementary aged students, an age group which is already vastly underrepresented in the collaborative programming literature (Tsan et al., 2018). This is a large gap in the literature because results for adults are not very generalizable to children who's cognitive, learning, and communication capabilities are vastly different. Further, because there is little research on the use of collaborative programming environments as an educational paradigm for elementary aged students, very little is understood regarding best practices for designing an educational collaborative programming environment for this age group. For instance, which would be more effective for elementary aged students: pair programming or side-by-side programming?

Theoretical Framework

Ryan and Deci's (1985) self-determination theory states that behavior is motivated by three psychological needs: "autonomy (the urge to control one's own life), competence (the urge to experience mastery), and psychological relatedness (the urge to interact with, be connected to, and care for others)" (National Academies of Sciences, 2018, p. 115). Using this theory, we theorize that students will be more likely to learn programming skills in environments where they perceive the greatest autonomy. This autonomy will then lead to greater intrinsic motivation to reach competence. Further, competence is more likely to occur when supported by a partner who can collaborate effectively.

Current Study

The current study examined the interface design of a collaborative computer science learning environment for elementary aged students. Our study consisted of three main objectives: 1) to see what collaborative programming paradigm elementary aged students preferred, side-by-side or pair programming; 2) to understand why students preferred one method over the other and what elements of each they liked and disliked; and 3) to explore ideas for how to effectively design a collaborative programming environment for elementary aged students. The current study's research questions reflect these objectives:

- **RQ1:** Which programming environment did students prefer: side-by-side or pair programming?
- **RQ2:** What challenges did students experience with pair and side-by-side programming?
- **RQ3:** What aspects of pair and side-by-side programming did the students like?
- **RQ4:** What are the students' suggested features for improving collaboration?

METHOD

Participants

15 students (33.33% female; 73.33% white) from two fifth-grade gifted classrooms participated in the study. One was a math class and one was an English language arts (ELA) class. These students participated in five computer science instructional days each spaced a week apart, for a total of five weeks.

Materials

Seven focus group questions (Table 1) directly aligned with our research questions were presented to students after five computer science instructional days. Students used NetsBlox (Figure 1), a block-based educational programming environment based on the visual programming language Snap!. Programming environments such as NetsBlox are commonly used in K-12 classrooms and have been shown to be an effective method to introduce coding concepts to elementary aged students (Broll, 2018). Additionally, NetsBlox enables side-by-side programming by having students login to the same virtual programming space from two separate computers. NetsBlox's side-by-side programming environment is synchronized, meaning partners can see each other's work in real-time.

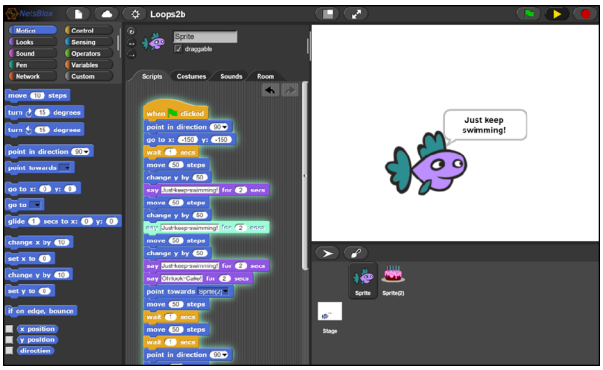

Figure 1: Screenshot of NetsBlox

Table 1: Focus Group Questions

Focus Group Question	Research Question Alignment
1) Which way did you prefer? When you pair programmed or when you worked side-by-side on two separate computers?	RQ1
2) What were some things that did not go well while you were pair programming?	RQ2
3) What were some things that did not go well while you were collaborating with your partner and programming on separate computers?	RQ2
4) What did you like about pair programming? Do you have any specific examples?	RQ3
5) What did you like about collaborating with your partner while programming on separate computers? Do you have any specific examples?	RQ3
6) Imagine you are designing a new version of NetsBlox with the goal of improving collaboration between two programmers. What would you change about NetsBlox to improve collaboration?	RQ4
7) Is there anything else that you would like to talk about or give feedback on?	

Procedure

Students first participated in five total instructional days each spaced a week apart. The classroom teacher was present each day and paired the students; the students were often paired in new ways each week. One of the researchers taught the computer science concepts each class. Additionally, the differences between side-by-side and pair programming were re-explained each class and during pair programming, students were told when to switch roles. Each class lasted 60 minutes and students alternated between side-by-side programming (Figure 2a) and pair programming (Figure 2b) each week. As we did not have access to additional instructional days, students were exposed to side-by-side programming one day more than pair programming. Also, the majority of the students were in both classes as they received gifted services for both math and ELA. Computer science topics covered in the curriculum included conditionals, conditional debugging, variables, loops, and pattern recognition. On the sixth week, students participated in focus groups. Focus groups were recorded and transcribed by the researcher. There were a total of four focus groups (see Table 2 for a breakdown of each group).

Analyses

Focus groups were transcribed using a combination of typing and the Google Docs voice typing tool. For research question one: Which programming environment did students prefer: side-by-side or pair programming? We took a general count of each student's answer to focus group question one. Research questions two, three, and four were analyzed using thematic analysis by one of the researchers. Thematic analysis is a qualitative method, selected for its flexibility and used to organize and describe a data set (Braun & Clark, 2006). These categories were checked for internal coherence, consistency, and distinction and then labeled based on overall themes. Thematic analyses were completed using the qualitative data coding software ATLAS.ti (ATLAS.ti, 2018).

Table 2: Focus Group Breakdown

Focus Group	Time	Gender Breakdown
1	26 Minutes	3 males, 1 female
2	28 Minutes	2 males, 2 females
3	24 Minutes	3 males, 1 female
4*	37 minutes	2 males, 1 female

*Focus group four was held a week later than the other groups as those students were absent from class the previous week.

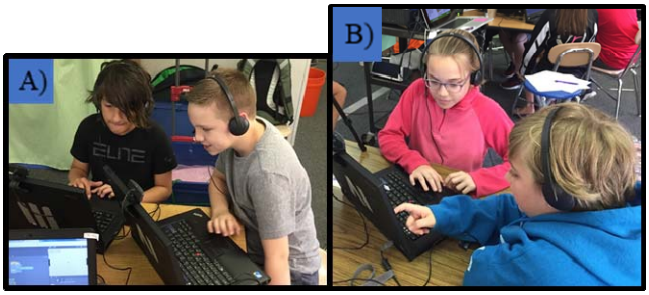

Figure 2: A) Side-by-side programming setup. B) Pair programming setup.

RESULTS

Research Question One: Which programming environment did students prefer: side-by-side or pair programming?

Twelve out of fifteen students stated they preferred side-by-side programming when asked the first focus group question, "Which way did you prefer?..."

Research Question Two: What challenges did students experience with pair and side-by-side programming?

From the thematic analyses, six challenges with pair programming and three challenges with side-by-side programming arose (Table 3). Specifically, for pair programming, it was difficult for some pairs to negotiate fair turn taking. For instance, students should have been the driver 50% of the time and the navigator 50% of the time; however, in some cases, a partner might monopolize the driver role making the split closer to 80% (driver)/ 20% (navigator), and vice versa (e.g., "maybe sometimes you would have a partner that would maybe just take over and do it all and then you're like, 'Hey can I have a turn.' 'No you already did it.'"). Contributing to unfair turn taking, students found it challenging to wait for their turn to be the driver as "it was hard to be patient," and both students "wanted to control it but… it can only be one at one time." Additionally, some partners had trouble communicating effectively. For instance, pairs would get into arguments over what they should do, or the navigator would warn the driver that, "…something is going to happen…," but their partner often did not, "…listen and they were the ones that are on the computer." Further, many students found the setup of pair programming to be "kind of cramped," and felt that pair programming offered significantly less hands-on experience as "…the navigator, they don't really get the hands-on experience." Lastly, students stated that the navigator was often not paying attention to what the driver was doing. For example, "sometimes when I would be the driver, my navigator wouldn't really listen, just dozing off … staring off into space and wasn't really paying

attention." Many students appeared to treat the navigator role as a break until it was their turn to be the driver again.

For side-by-side programming, students brought up technical issues such as "the lag makes it slow…," or "sometimes when I would edit something, it wouldn't show up on their screen." The students also stated that they often worked a little too independently which led to poor coordination of work. For example, "… if you were working on the same area of code, it would be kind of hard because you're both kind of doing your own thing, so it just kind of jumbles up." Lastly, it was problematic when students worked on different components of the code because "you can't really see what the other person is doing…"

Table 3: Research Question Two Results: Challenges with Pair and Side-by-Side Programming

	Theme (# Comments)	Number of Comments
Pair	Problems with Turn Taking	6
	Challenging to Wait for Turn to Be the Driver	5
	Poor Communication: Arguing and Not Listening to Partner	5
	Physical Setup Was Too Cramped	4
	Lack of Hands-on Experience	3
	Navigator Not Paying Attention	3
Side-By-Side	Technical Issues	8
	Working Independently: Poor Coordination between Partners	5
	Can't See What Partner is Doing	8

Research Question Three: What aspects of pair and side-by-side programming did the students like?

From the thematic analyses, three positives with pair programming and five positives with side-by-side programming arose (see Table 4). For pair programming, some students felt they were able to learn more during pair programming as "you can learn from what your partner is doing" and your partner can help correct any mistakes you make. Additionally, many students stated that, "it's easier to see what the other person is doing" compared to side-by-side programming. And lastly, it was much easier to "catch someone's mistake" while pair programming compared to side-by-side programming.

Table 4: Research Question Three Results: Positives of Pair and Side-by-Side Programming

	Theme (# Comments)	Number of Comments
Pair	Learn from More from Watching Partner	7
	Easier to See What Partner is Doing	6
	Easier to Find Mistakes	6
Side-by-Side	Efficiency	8
	More Independence and Control	8
	Learned More	4
	Less Cramped	3
	More Hands-on Experience	3

For side-by-side programming, students felt they were able to get work done more efficiently "because one person can work on one thing and another could work on the other." Students also liked that side-by-side programming offered more independence and control as students did not have to wait their turn to enact their ideas (e.g., "…felt like when I had the side-by-side I could always, like if I had an idea, I could do it then"). While some students felt they learned more from pair programming, others felt side-by-side enabled them to "learn more because you had more of a chance to experiment and learn more about what each thing did." Students also stated that side-by-side programming's setup was much less cramped and that they got significantly more hands-on experience when side-by-side programming.

Research Question Four: What are the students' suggested features for improving collaboration?

Students gave several suggestions for improving software interfaces that would support side-by-side and pair programming. Students' top suggestions for side-by side programming were as follows. Add an instant messaging function (6 comments) to communicate with a partner (e.g., "you could have like a message box or something on the side…"). Students also offered design ideas that looked a lot like Google Docs features which the students use regularly at school (3 comments). These features included having an indicator to show exactly where a partner is working in the code (e.g., "maybe like on Google Docs like if you're on part of like the document, it shows like a little color to [inaudible] that you're there"); have an option to make suggestions rather than directly edit the code, and be able to send partner sections of code and have them fix it and then send it back (e.g., "if there were suggestions that [inaudible] then you can send it to them like Google Docs. It would say that like they made a suggestion you could either undo it or leave it").

For pair programming, students' suggestions were as follows. First, some students suggested a timer to indicate when it is time to switch. However, not all students liked the idea of a timer, "like if it showed the timer the whole time I would be rushed to get done work I wanted to get done before I had to switch back." Students also suggested changing the physical setup so it would be less cramped (3 comments). Some mechanisms for this change included having a bigger screen, having two mice and keyboards and have the navigator not look directly over the driver's shoulder (e.g., "like make the driver better by putting in the navigator...not like right behind their shoulder because it like puts pressure on you").

DISCUSSION

Participants overwhelmingly preferred side-by-side programming due to feeling they had more autonomy, were able to get work done more efficiently, learned more, and got more hands-on experience compared to pair programming. Additionally, side-by-side programming still enabled students to collaborate by having them simultaneously work on the same program from two workstations. The benefits of side-by-side programming align with self-determination theory which states that behavior is motivated by autonomy, competence, and psychological relatedness (Deci & Ryan, 1985). Specifically, the benefits of side-by-side programming themes—more independence and control, efficiency, learning more, and more hands-on experience—directly map onto two of the three factors necessary for self-determination: autonomy and competence. However, even though side-by-side programming enhanced autonomy, there may be other

downsides, as demonstrated by student comments (e.g., lag, poor coordination). Additionally, in terms of competence, the theme of 'learned more' arose for both pair and side-by-side programming; however, the reasoning for why they learned more differed between the two.

Students suggested pair programming enabled greater communication between partners allowing for more partner-to-partner learning while side-by-side enabled learning via hands-on experience. Hands-on experience even became an additional theme for the benefits of side-by-side programming while the lack of hands-on experience was one of the issue themes for pair programming. Additionally, the independence gained in side-by-side programming seemed to mitigate some of the issues found in pair programming. For instance, during pair programming, students found it hard to wait for their turn, leading to problems with fair turn-taking as most students wanted to be the driver. High tensions over who controlled the code led to poor communication methods such as arguing or the driver ignoring navigator suggestions. Many navigators chose to not pay attention at all because they did not think their role was important, and when they offered suggestions, the driver often did not act on these suggestions. The complaints for pair programming align with self-determination theory as students reported less autonomy, hands-on experience (competence), and often did not experience good cohesion with their partner (psychological relatedness). Past research with self-determination theory suggests that when students do not meet these three psychological needs, they are more likely to experience decreased intrinsic motivation and disengagement (Deci and Ryan, 1985).

Design Recommendations

These findings indicate that side-by-side programming may be the best paradigm to teach elementary aged students programming skills; however, more research is necessary to see if these findings are replicable, especially since this study had a small sample size and used a less representative sample: students from a gifted classroom. Additionally, future side-by-side programming environments should consider some of the design recommendations students suggested. First, technical issues such as lag should be prevented. However, there may be cases where uncontrollable factors, such as a slow internet connection, cannot be prevented, meaning side-by-side programming environments should also be designed with these limitations in mind. Additionally, we believe features such as text or video-based messaging for remote use, an indicator to show where a partner is in the code, and the ability to make suggestions would be beneficial additions to a side-by-side programming environment. Future research should empirically test the utility of each design feature on their ability to improve both learning and collaboration.

Conclusions

Although past research shows pair programming as beneficial to learning, these finding are likely not generalizable to elementary aged students because much of the research was done with older students, and elementary aged students have very different cognitive capacities. Our findings help bridge several gaps in the collaborative programming literature by examining collaborative programming with elementary aged students and comparing pair versus side-by-side programming. Additionally, we provided data to inform the design of new collaborative programming environments for elementary aged students.

ACKNOWLEDGEMENTS

We would like to thank our cooperating teachers and members of the STEM Cyberlearning Lab at the Friday Institute. This material is based upon work supported by the National Science Foundation under Grant No. DRL-1721000.

Reference

Ashcraft, C., Eger, E., & Friend, M. (2012). Girls in IT: The facts. *National Center for Women & IT*. Boulder, CO.

ATLAS.ti (Version 8) [Computer software]. (2018).

Baser, M. (2013). Attitude, gender and achievement in computer programming. *Online Submission*, *14*, 248-255.

Broll, B. (2018). *Collaborative educational environment design for accessible distributed computing* (Doctoral dissertation). Retrieved from Vanderbilt Library.

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology*, *3*, 77-101.

Bureau of Labor Statistics (2018). *Computer and information technology occupations*. Retrieved from https://www.bls.gov/ooh/computer-and-information-technology/home.htm

Deci, E., & Ryan, R. M. (1985). *Intrinsic motivation and self-determination in human behavior*. Springer Science & Business Media.

Denner, J., Werner, L., Campe, S., & Ortiz, E. (2014). Pair programming: Under what conditions is it advantageous for middle school students?. *Journal of Research on Technology in Education*, *46*, 277-296.

Dingsøyr, T., & Dybå, T. (2012). Team effectiveness in software development: Human and cooperative aspects in team effectiveness models and priorities for future studies. In *International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)* (pp. 27–29).

Google, & Gallup. (2015). *Searching for computer science: Access and barriers in U.S. K-12 education*: Google for Education.

Hill, C., Dwyer, H. A., Martinez, T., Harlow, D., & Franklin, D. (2015, February). Floors and Flexibility: Designing a programming environment for 4th-6th grade classrooms. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (pp. 546-551). ACM.

K-12 Computer Science Framework Steering Committee. (2016). *K-12 computer science framework*. Retrieved from https://k12cs.org

McDowell, C., Werner, L., Bullock, H. E., & Fernald, J. (2006). Pair programming improves student retention, confidence, and program quality. Communications of the ACM, 49(8), 90-95.

Nawrocki, J. R., Jasiński, M., Olek, Ł., & Lange, B. (2005, November). Pair programming vs. side-by-side programming. In *European Conference on Software Process Improvement* (pp. 28-38). Springer, Berlin, Heidelberg.

Papadakis, S. (2018). Is pair programming more effective than solo programming for secondary education novice programmers?: A case study. *International Journal of Web-Based Learning and Teaching Technologies*, *13*, 1-16.

Salleh, N., Mendes, E., & Grundy, J. (2011). Empirical studies of pair programming for CS/SE teaching in higher education: A systematic literature review. *Transactions on Software Engineering*, *37*, 509–525

Tsan, J., Rodríguez, F. J., Boyer, K. E., & Lynch, C. (2018, February). I think we should...: Analyzing elementary students' collaborative processes for giving and taking suggestions. *In Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (pp. 622-627). ACM.

Williams, L., Wiebe, E., Yang, K., Ferzli, M., & Miller, C. (2002). In support of pair programming in the introductory computer science course. *Computer Science Education*, *12*, 197-212.

Wilson, A., Hainey, T., & Connolly, T. (2012, October). Evaluation of computer games developed by primary school children to gauge understanding of programming concepts. In *6th European conference on games-based learning (ECGBL)* (pp. 4-5).