A Comparison of Two Pair Programming Configurations for Upper Elementary Students

Jennifer Tsan[1], Jessica Vandenberg[1], Zarifa Zakaria[1], Joseph B. Wiggins[2], Alexander R. Webber[2], Amanda Bradbury[1], Collin Lynch[1], Eric Wiebe[1], Kristy Elizabeth Boyer[2]
jtsan@ncsu.edu,jvanden2@ncsu.edu,zzakari@ncsu.edu,jbwiggi3@ufl.edu,alexwebber@ufl.edu
aebradbu@ncsu.edu,cflynch@ncsu.edu,wiebe@ncsu.edu,keboyer@ufl.edu
[1]North Carolina State University, Raleigh, North Carolina
[2]University of Florida, Gainesville, Florida

ABSTRACT

As computer science education opportunities for elementary students (grades K-5) are expanding, there is growing interest in using pair programming with these students. However, previous research findings do not fully support its use with younger learners, and some researchers have begun to examine whether introducing a second computer with a shared coding workspace can provide important benefits. This experience report describes a series of classroom activities in the 4th and 5th grades (ages 9-11 years old) with two different pair programming configurations: one-computer pair programming, in which both students share a keyboard, mouse, and monitor; and two-computer pair programming, in which each student has a separate computer but coding workspaces are synchronized over the web. In both cases the students sat next to each other and engaged in face-to-face conversation. We found that students largely preferred two-computer pair programming over one-computer pair programming. We conducted focus groups and transcribed collaborative dialogues to gain more insight into this preference. We learned that students felt more independence in two-computer pair programming, although they struggled with coordinating their edits with their partner. In one-computer pair programming, students reported not wanting to wait for their turn to drive, but feeling as though they communicated more with their partner. Both configurations can be productive for students, but the tradeoffs described in this experience report are important for CS educators and researchers to consider when determining which collaborative configuration to use in each K-5 classroom context.

CCS CONCEPTS

• **Social and professional topics → K-12 education**; • **Applied computing → Collaborative learning**.

KEYWORDS

Pair programming, K-5, collaborative learning

SIGCSE '20, March 11–14, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6793-6/20/03…$15.00
https://doi.org/10.1145/3328778.3366941

ACM Reference Format:
Jennifer Tsan[1], Jessica Vandenberg[1], Zarifa Zakaria[1], Joseph B. Wiggins[2], Alexander R. Webber[2], Amanda Bradbury[1], Collin Lynch[1], Eric Wiebe[1], Kristy Elizabeth Boyer[2]. 2020. A Comparison of Two Pair Programming Configurations for Upper Elementary Students. In *The 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20), March 11–14, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3328778.3366941

1 INTRODUCTION

Pair programming, a configuration in which two programmers work on one computer while taking turns at the controls, has been used in introductory computer science courses and in the programming industry for over two decades [3, 4]. Typically, in this configuration, the person controlling the keyboard and mouse acts as the *driver* making all code changes, while the other person acts as the *navigator* and is tasked with planning ahead and looking for mistakes [30]. Both programmers are expected to engage with each other continuously as they work, to solve problems collaboratively, and to switch roles after a set amount of time or a set portion of the task has been completed. We refer to this configuration as *one-computer pair programming* because both programmers share one computer.

There is a growing interest in using pair programming with younger students to build strong, early foundations in computer science (e.g., [7, 10, 27]). Although one-computer pair programming has been shown to be beneficial for older novices, studies indicate that this collaborative configuration may not be suitable for younger learners [20, 27, 28]. This may be due in part to the fact that cognitive, learning, and communication capabilities essential to effective pair programming are still developing for students at this age and show wide student-student variance [17, 18, 24].

In our work in elementary school classrooms, we initially used one-computer pair programming. However, we noticed the students sometimes struggled to regulate control of the keyboard and mouse, leading to conflicts. We turned to the literature where we found examples of alternative pair-programming configurations. CS Education researchers have investigated collaborative configurations such as "intermittent collaboration" [19] or "side-by-side programming" [8, 22], in which two programmers work on independent computers and can work on separate coding assignments or divide-and-conquer joint tasks as they deem appropriate. Multi-user input with a shared computer has also been explored, such as giving each programmer a separate mouse or other independent input device [12, 16, 26].

Inspired by that work, we decided to investigate an alternative to one-computer pair programming by providing each child with a computer and synchronizing their workspaces over the web using the NetsBlox programming environment [6]. In this paper will refer to this configuration as *two-computer pair programming*. While prior researchers have sometimes referred to this model using other terms such as "intermittent collaboration" [19], or "side-by-side programming" [22] we believe that *two-computer pair programming* best describes our approach. In our view "collaboration" can encompass any number of participants, but our work focuses on pairs. Additionally, "side-by-side programming" can imply any type of interaction including work on separate projects, but our students share a single task. This structure leads to dynamics that are similar to a one-computer condition with students communicating about their work, offering suggestions, and exchanging control of code, even though the driver/navigator roles are not strictly enforced.

We use the term *two-computer pair programming* to refer to contexts in which each of two programmers has a computer and 1) each has full, parallel input control and viewing; 2) they are in close physical proximity so that they can talk and gesture, including pointing at each other's computer screens; 3) they work in a synchronized, shared development environment (Figure 1).

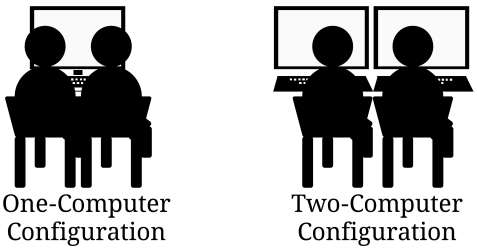


Figure 1: Pair Programming Configurations [32]

We have used both a one-computer and two-computer configuration with 4th-5th grade students in the southeastern US over the past two years as part of a project to integrate CS into elementary school classrooms. This experience report describes the trade-offs that we observed between those two configurations. For brevity, we refer to one-computer pair programming as ***1C*** and two-computer pair programming as ***2C***. We observed that students generally preferred 2C because they felt that it gave them more hands-on experience and independence. Some students felt that they learned more in 1C while others felt that they learned more in 2C. Students also found that it was easier to coordinate and keep track of what their partners were working on in 1C. The lessons learned from this experience report can inform future classroom efforts involving pair programming with upper elementary learners.

2 RELATED WORK

Pair programming began as an industry practice and has since been shown to improve adult programmers' efficiency, satisfaction, and confidence [30]. Additionally, Nosek found that developers who used the pair programming configuration created code that was more readable and functional than individual programmers [23]. Not long after pair programming was introduced, postsecondary faculty began exploring this configuration in introductory computer science classrooms. Studies at the undergraduate level showed that students who engaged in pair programming had a higher rate of earning a C or better in their classes, and performed better on tests and projects than students that worked individually [31]. Course retention and major retention have also improved with pair programming, along with confidence, especially for women [29].

Compared to a sizeable body of research findings on the effectiveness of pair programming with adult developers including undergraduate students, the study of pair programming with K-12 students began more recently. Denner et. al. found that middle school students who engaged in pair programming did better in computational thinking post-tests and had a greater increase in Alice knowledge than individual programmers [10]. In a sixth grade coding camp, Lewis collected collaboration data to understand which collaborative configuration was better for younger students solving computer science problems: pair programming or "intermittent collaboration" [19]. In the pair programming condition, the students were required to switch roles every five minutes. In the "intermittent collaboration" condition, participants worked next to each other without a synchronized workspace and were required to discuss their progress and challenges every five minutes, and ask each other for help before they could ask an instructor. Lewis found that students in the "intermittent collaboration" condition had more positive responses to continuing Scratch and programming in the future, and completed activities in less time.

Side-by-side programming is often conceptualized as two programmers working in close proximity to each other, on their own computers, but with the ability to complete shared tasks [8, 22, 25]. Cockburn [8] surmises that side-by-side programming is an appropriate alternative to pair programming, which often leaves programmers feeling that they are being monitored and with little time to work on other projects. Side-by-side programmers typically cease independent work to talk with their partner about work strategies or next steps, to share knowledge, to debug the code, or to combine their work [25].

Distributed variants of side-by-side programming have also been proposed [11]. In this configuration, programmers can work remotely, but each has an *awareness computer* that displays the work their partner is completing in real-time. Programmers working in this configuration structure their time in varied ways; they work independently without talking, they work on sub-tasks and speak when needed, they pair program, one programs while the other browses, and they jointly browse. The authors maintain the superiority of distributed side-by-side programming because of the flexibility with which the programmers could engage in joint and independent work.

In our examination of the literature for guidance on how to support pair programming with elementary students, we did not find any studies that examined two-computer pair programming with this young student population. This experience report aims to shed light on the tradeoffs involved between the one-computer and two-computer programming configurations.

School	White	Latinx	Black/Afr. Am.	Asian	Multi.	NSLP*
Clark	51	27	14	3	5	44
Frederick	58	19	16	1	4	42
Greg	64	3	22	3	8	63

Table 1: Demographics of the schools by percentage. *National School Lunch Program - free/reduced lunch

3 CLASSROOM EXPERIENCES

In this experience report, we detail a series of CS learning opportunities that we brought to 4th and 5th grade students. In these learning opportunities we had students try 1C and 2C pair programming as they developed code in NetsBlox [6], a block-based programming platform that permits students to invite collaborators to help create content. The code is synchronized over the web as each coder makes edits.

The first classroom experience (State 1) used a science-based curriculum. The remaining two classroom experiences and focus groups (State 2) used a computational thinking/computer science curriculum. The demographics of the students are listed in Table 1. The states and names of elementary schools have been anonymized. Informed consent was obtained for all students whose data are analyzed here according to an IRB-approved protocol. For each classroom experience, we made observations and developed questions based on the observations. This required us to take on different annotation schemes to analyze the data from each experience.

3.1 Classroom Experience 1 - Science

3.1.1 Curriculum. The curriculum for this classroom experience was designed and taught by the authors. We collaborated closely with our partner teacher to plan, create, and revise these tasks. It covered not only the computational thinking concepts, but also science concepts surrounding the classroom's science focus prior to our classroom experience, pollination. The experience spanned a three-day period in a 4th grade science class, each lesson designed to last 45-50 minutes. The learners in these classes had participated with our team in a previous learning experience within the same semester, during which they learned fundamental block-based coding concepts, including message passing, conditionals, and loops.

On day 1, we led an overview of a computer science concept, nested conditionals, which built upon the lessons students had completed in the previous experience. On days 2 and 3, students worked in pairs to complete learning tasks focused on the science concept of pollination using nested conditionals. Student pairs were created by the teacher prior to our classroom experience (randomly or by who would work best together). The morning class first experienced the 1C configuration and then the 2C configuration on the following day. The afternoon class experienced the conditions in the opposite order.

3.1.2 Demographics. This experience was offered at a public school with 38 fourth grade students. Participants included 16 girls and 22 boys between the ages of 9 and 11. The students attended one of two science classes (18 in morning, 19 in afternoon), each instructed by the same teacher. The classes were both 50-minute sessions.

3.1.3 Observations. The 4th and 5th grade students at Greg Elementary had previous experience with our group and were excited to work with us again. As they were already familiar with the block-based language, we gave them a minimal overview of the material. Our student population was split between two distinct class groups, one in the morning and one in the afternoon with no participant overlap. We observed that students in the 1C configuration for both classes experienced more interpersonal conflict than the days in which they worked in 2C. However, we also noticed some students in the 2C configuration were not contributing to the solution and did not seem fully engaged in the learning process.

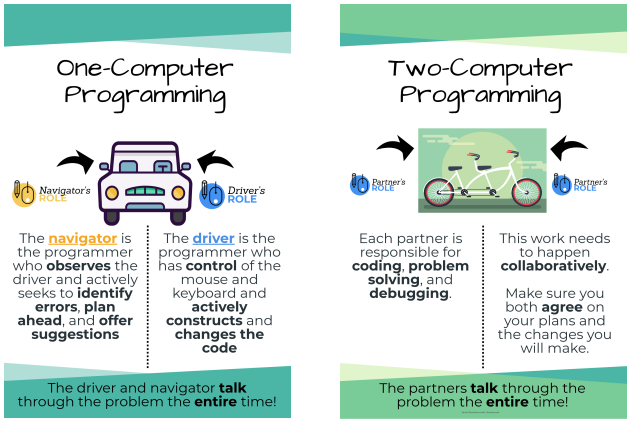
We were interested in how conversations unfolded in the two different configurations since this is an indicator of the depth of collaboration and learning. A preliminary analysis focused on the ways students asked questions of each other, labeling eight question types: DESCRIPTION (*e.g., What's the stage?*), METHOD (*e.g., How do we get out of this?*), EXPLANATION (*e.g., Why do you have to go to motion?*), RATIONALE (*e.g., What are you doing?*), COMPARISON (*e.g., Should we shorten it up?*), PREFERENCE (*e.g., Do you want to read it better?*), BINARY (*e.g., So red or yellow?*), and STATUS (*e.g., What did I do?!*). These question types are taken from a widely used question taxonomy for learning [14].

To label these questions, two researchers independently reviewed two videos in order to find occurrences of question-asking behavior. Each researcher transcribed each instance, which included the start and end time of each question episode and the reason for including the event as a question. Disagreements in transcription were resolved through discussion and iterative refinement. For the beginning and end time of each question, we added 5 seconds before and after the timestamps labeled for each question. For example, if the question asking behavior episode began at the timestamp of 00:55 and ended at 01:00, we stamped the episode to start at 00:50 and end at 01:05. After the instances of question asking were extracted, the two researchers discussed each instance. Cases of disagreement were resolved by either reaching consensus that a question-asking episode had occurred, or removing that instance if it was not agreed upon. Once the discussion period for each transcribed question episode concluded, we completed the same process for annotating each question.

We found no significant differences in the number of any question type across 1C and 2C configurations. The most frequent question types were EXPLANATION questions and STATUS questions. An EXPLANATION question asks for the clarification of the causes, context, and consequences of facts, such as "Why do you have to go to motion?". A STATUS question refers to the current condition of the program as produced by a partner or the self, such as "What did I do?!" Although further analysis is needed, these initial results indicate that there may be no significant differences in question-asking frequency across 1C and 2C configurations.

3.2 Classroom Experience 2 - CT 1

3.2.1 Curriculum. The curriculum for this classroom experience was designed by the authors and taught by the school media center teacher. Each lesson was intended to last an hour.Students were first taught about *programming* and *algorithms* before they were introduced to the block-based programming environment. On day 2,



(a) 1C Poster **(b) 2C Poster**

Figure 2: 1C and 2C Posters.

students were introduced to the categories of blocks and practiced their use. Day 3's lesson consisted of three activities on conditionals and conditional trees, culminating in an activity in which students created their own idea for a program that would require conditionals, draw the conditional tree, and then program it. On day 4, students learned about repeating patterns and loops. They worked in pairs to identify a program they could write that needed loops and then implemented it. On day 5, students learned about taking in user input and adapting program behavior based on input, while on day 6 they learned about broadcasting and receiving. Finally, on day 7, students created a game that brought together many of the concepts covered previously.

At Clark Elementary, the students were in the class for 45 minutes; therefore, the lessons had to be reduced to fit in that time slot. The course was taught by the media center specialist who had some experience teaching block-based programming. In addition to shortening the lessons, the teacher decided to cover conditionals and loops for two days each after seeing the students struggle with the concepts during the first day of each lesson. The final lesson the teacher taught was loops.

As part of this curriculum, the students were taught about both 1C and 2C pair programming. They were introduced to the roles and responsibilities of the driver and navigator and they were taught about the importance of talking through their decision making process. The teacher shared digital posters for both 1C (Figure 2a) and 2C (Figure 2b) and reminded students of how they should work in each configuration. She did not, however, remind the students to switch roles in 1C. The first and third programming days were 1C and the second and fourth were 2C.

3.2.2 Demographics. At Clark Elementary, the three 5th-grade classrooms in which the experiences occurred contained approximately 28 students each and were representative of the overall population of the school. 68 students participated in the study (29 girls, 39 boys). The students were paired by their homeroom teachers based on their prior collaborative behaviors; the pairs remained the same over the course of the entire experience.

3.2.3 Observations. The students at Clark Elementary had minimal experience with coding and were excited to spend their media center time engaged in this opportunity. After receiving instruction from the teacher, we noticed that some of the students jumped right into the coding task, whereas others took time to plan. Moreover, we observed that the ways the students talked to each other seemed to change depending on whether they were using one computer or two. Based on these experiences, we wanted to explore more deeply how students interact in 1C and 2C pair programming.

All 68 consenting students across the three classrooms were video recorded as they worked together on one or two computers. Videos were transcribed verbatim of each day the students programmed. We annotated the transcripts according to which task phase they were in (planning, monitoring, or evaluating) and how their language indicated they were working together (collaborative, agreement, tutoring, disagreement, confusion, or individualistic). We analyzed annotated transcripts from four pairs (four girls and four boys) in which the students participated in all four programming lessons, described previously. Moreover, we calculated frequencies of each type of talk each student used each day.

In general, for both conditions, the analysis found that students rarely planned before starting their work, nor did they reflect on their work after completing a task. Regarding differences in 1C and 2C, we noticed there was more disagreement-annotated conversations on 1C days and more collaboration-annotated conversations on 2C days. Of note, when the task became more challenging—on the final two days—the conversations shifted to become more individualistic, and to reflect more disagreement and confusion, regardless of whether the students were in 1C or 2C.

3.3 Classroom Experience 3 - CT 2

3.3.1 Curriculum. We modified the original curriculum to fit the requirements needed for the 5th grade classes at Frederick Elementary. On day 1, students learned conditionals and completed two programs where the sprites acted differently based on certain conditions. On the next day, students learned about variables and completing debugging activities. On day 3, students learned about loops and on day 4 students were given starter code and instructions for programming four games, and chose which one(s) they wanted to make. On the fifth day of instruction, we conducted focus groups with the students to explore how they felt about 1C and 2C pair programming. There were three to four students in the focus groups. We divided them up randomly. The questions we asked were: *1) Which way did you prefer? 1C or 2C? 2) What were some things that did not go well during 1C? 3) What were some things that did not go well during 2C? 4) Do you feel you learned more of the programming concepts using one setup over the other? Why? 5) What did you like about 1C? Do you have any specific examples? 6) What did you like about 2C? Do you have any specific examples?*

One of the authors taught the class, and each lesson lasted one hour. Similar to the Clark Elementary curriculum described previously, the students were taught about both 1C and 2C pair programming. They were introduced to the roles and responsibilities of the driver and navigator and they were taught about the importance of talking through their decision-making process. We had printed versions of the posters that were placed in front of the students. At

Config.	Challenges
1C	1. Problems with Turn Taking 2. Challenging to Wait for Turn to be the Driver 3. Poor Communication: Arguing and Not Listening to Partner 4. Physical Setup Was Too Cramped 5. Lack of Hands-on Experience 6. Navigator Not Paying Attention
2C	1. Technical Issues 2. Working Independently: Poor Coordination between Partners 3. Can't See What Partner is Doing
Config.	Benefits
1C	1. Learn More from Watching Partner 2. Easier to See What Partner is Doing 3. Easier to Find Mistakes
2C	1. Chunking 2. More Independence and Control 3. Learned More 4. Less Cramped 5. More Hands-on Experience

Table 2: Benefits and Challenges of 1C and 2C.

the beginning of each class, we reminded the students about 1C and 2C pair programming. Another difference between this implementation and the previous one is that during 1C days, we set a timer for the students and announced when it was time to switch.

3.3.2 Demographics. Frederick had 11 participants, 33.33% were girls. These students participated in the five-day curriculum with each lesson spaced a week apart. The students were randomly paired by their teacher every lesson.

3.3.3 Observations. The Academically or Intellectually Gifted (AIG) students were excited about the programming experience where the teacher would quiz them on the roles of driver and navigator (in 1C) or the function of certain blocks. The children would talk among themselves about their plans for the day, deciding who would be responsible for which component of the task or how they might change what they had previously completed. The students were enthusiastic but we were unsure of the quality of their collaborative conversations. This led us to look at how the kids critically talked to each other, how they were positive towards each other, and how the students pushed each other's thinking to higher levels. Literature shows that conversation is considered productive when individuals are critical of each others' propositions by challenging and explaining themselves along with offering alternative ideas (Exploratory conversation) [13], and less productive either when students uncritically converse with intention to avoid conflict (Cumulative conversation) or when they tend to have unsolved disagreements (Disputational conversation) [21].

We used video recordings of student conversations during pair programming to examine the types of conversations that they participated in throughout the activities. We watched 20 minutes of concurrent video with transcripts of students' collaboration to qualitatively annotate the three categories of conversation mentioned above. Video recordings were divided into 120 ten-second intervals and we annotated each interval exclusively to one of the three categories (Exploratory, Cumulative, or Disputational).

We found that overall, students used Cumulative conversation more than the other two categories in both 1C and 2C conditions [32]. There were no significant quantitative differences in types of conversation between 1C and 2C, however, we had three interesting qualitative observations: First, in 1C, challenging ideas resided primarily with the driver while the navigator was left defending their ideas. Second, instances of Exploratory conversation were often preceded or followed by Disputational in 1C, whereas, it transitioned into Cumulative conversation in 2C. Finally, evidence suggested that 2C has the risk of dissipating the collaborative relationship by turning into a cooperative one, where students worked in parallel rather than focused on the same immediate task [9]. Nonetheless, even when the students worked more cooperatively in 2C, partners had consulted each other regularly and had a balanced opportunities to challenge and explain their thoughts to each other.

4 EXAMINING STUDENTS' EXPERIENCES

One of our goals in these classroom experiences was to better understand how the students felt about 1C and 2C. We asked the students at Frederick Elementary about what they liked and disliked about each experience (described in Section 3.3). When asked which method they preferred, twelve out of fifteen students stated they preferred 2C over 1C [5].

We then asked the students about challenges they ran into while using the methods (Table 2). For 1C, students stated that a partner could monopolize the driver role (which we observed on many occasions). The students preferred the driver over the navigator role and did not enjoy waiting their turn to be the driver again. They found that in 1C, communication was less effective because of arguing and partners not listening (which we also documented). Additionally, there were fewer opportunities for hands-on experience in 1C. Since each student had a computer, the students felt that there was not enough space to sit comfortably. The students stated that the navigator was not always attentive to what the driver was doing. For 2C, students mostly spoke about technical issues such as latency issues with NetsBlox in which the programs did not synchronize quickly. As the students were not able to easily see their partners' actions in real time, these issues often led to students working in parallel and not fully coordinating their work.

Finally, we asked them about positive aspects of each method (Table 2). For 1C pair programming, some students felt they were able to learn more because they could see what their partners were doing, and as a result, they could see their partners' mistakes more easily. For 2C pair programming, the students felt that they were able to chunk the work into sub-tasks and work in parallel. They felt that they had more independence and control in 2C. Additionally, since there was only one computer per pair, it was less cramped than in 2C. On the other hand, the students believed that 2C allowed for more hands-on experience. Additionally, there were some students that thought they learned more in 2C than 1C.

5 DISCUSSION AND TAKEAWAYS

After observing students in the three schools try both 1C and 2C pair programming, we began to reflect on how those experiences went. Overall, the students expressed greater satisfaction with 2C, but they were able to identify benefits and challenges of both. We summarize a set of takeaways for practitioners to consider when deciding between using 1C and 2C pair programming.

Independence. Students mentioned that they could work more efficiently in 2C with independence and control over the task. While working independently may be desirable in some contexts, we noticed that the students often worked on different parts of their program while talking to each other about the code, which could be considered cooperative work instead of collaborative work [15]. Although the students' process may resemble cooperative work, Classroom Experience 1's question-asking analysis suggests that students may ask a similar number of questions regardless of whether they were in the 1C or 2C configuration, indicating that meaningful dialogue is still occurring. Additionally, as suggested by the analysis of Computational Thinking 2, students had an equal number of Cumulative, Exploratory or Disputational statements in 1C and 2C. Therefore, we recommend the use of 2C when practitioners would like to provide more agency to their students and when there is less concern that imbalance of ability or motivation would lead to one student doing a majority of the work.

Coordination. We also noticed an interesting pattern as students negotiated over the task. The students felt there was often poor coordination between partners when using 2C. One student stated, "you're both kind of doing your own thing, so it just kind of jumbles up." Moreover, in 1C, students found it hard to wait for their turn, leading to problems with fair turn-taking as most students wanted to be the driver. Tension over who controlled the code led to poor communication methods such as arguing or the driver ignoring navigator suggestions. Many navigators chose not to pay attention to the driver's work because they did not think their role was important, and when they offered suggestions, the driver often ignored them. We have observed this pattern many times where some students remain the driver the entire session. Practitioners who use either method should be aware of the need to scaffold effective communication practices, specifically turn-taking in 1C and task coordination in 2C.

Learning. Students stated that they "learned more" in both 1C and 2C; however, the reasons they offered as explanation differed between the two configurations. Students suggested 1C allowed for more partner-to-partner learning ("you can learn from what your partner is doing"), whereas 2C enabled learning via more hands-on experience. Students mentioned preferring 2C as they found this configuration provided more agency than 1C ("learn more because you had more of a chance to experiment and learn more about what each thing did."). This is not surprising given that the setting of 1C permits only the driver to control the input devices. Although we do not have evidence of students learning more in either condition, students' perception of how much they learn can affect their self-efficacy [2]. Practitioners should monitor this affective dimension as different students pairs may respond differently to 1C and 2C.

Conversations. There were important, though subtle, differences in the patterns of conversations between the two conditions. We noticed that conversations turned into unresolved disagreements more in 1C than in 2C. In addition, we also observed that as the difficulty of tasks rose, the students disagreed more. Practitioners should be alert that when the task difficulty rises, they need to be attuned to increases in disagreement, especially in 1C.

Chunking. Some students expressed that they enjoy the 2C configuration because they could divide the tasks and complete the subtasks in parallel. One student stated that this was "because one person can work on one thing and another could work on the other." This is an important practice in computer science, "At any grade level, students should be able to break problems down into their component parts" [1]. Practitioners can support this by teaching the students how to chunk their tasks in 2C. However, they also need to be aware that a higher level of coordinating communication will be required of the students.

Other Logistical Considerations. *Physical comfort* of students is an important consideration. In our experience, some desks were too full with one device per student, while other desks had plenty of room. *Device size* is also important: devices with smaller screens create scenarios in which students had to sit quite close to each other or staggered so the navigator sat looking over the driver's shoulder. 2C would require classrooms to have a sufficient *number of devices* for each student to have an individual computer.

6 CONCLUSIONS

When teaching computer science, it is important to consider the social and learning needs of all students and choose appropriate methods to support learning. Pair programming is often used to facilitate communication and collaboration in classrooms, although its effectiveness with different age groups is still under investigation. Our observations from classroom experiences are consistent with the suggested conclusions from prior research, that elementary-aged students have vastly different cognitive and communication capabilities than adults, and that these differences need to be taken into account when designing pair programming activities.

To help inform solutions that address these differences, this experience report has compared a one-computer (1C) and two-computer (2C) configuration for pair programming with these young students. Among other findings, students felt that they had stronger communication with their partners while using 1C, whereas students felt they were getting better hands-on experience while using 2C. Although students reported better coordination while working on the same task in 1C, they frequently argued over who should be in control of the computer and, once in control, would be resistant to considering input from their partner. While using 2C, students feel more efficient, dividing and conquering parts of the code, though disconnected from what their partner was doing.

We are only at the beginning of understanding elementary-aged students' perceptions and experiences during pair programming. While our experiences point to benefits and challenges of both configurations, it is critical that we observe how partner collaboration unfolds out over longer periods of time. The CS Education community needs to continue to investigate how students come to develop a sense of identity toward computer science, the language that unfolds between pair programmers, as well as other factors that drive classroom pedagogy.

7 ACKNOWLEDGMENTS

This work is supported by the National Science Foundation through the grant DRL-1721160. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Julie Alano, Derek Babb, Julia Bell, Tiara Booker-Dwyer, Leigh Ann DeLyser, Caitlin McMunn Dooley, Diana Franklin, Dan Frost, Mark A. Gruwell, Maya Israel, and et al. 2018. K12 Computer Science Framework. (2018). https://k12cs.org/

[2] Albert Bandura. 1993. Perceived self-efficacy in cognitive development and functioning. *Educational psychologist* 28, 2 (1993), 117–148.

[3] Kent Beck. 1998. Extreme Programming: A Humanistic Discipline of Software Development. In *International Conference on Fundamental Approaches to Software Engineering*. Springer, 1–6.

[4] Kent Beck. 1999. Embracing Change with Extreme Programming. *Computer* 32, 10 (1999), 70–77.

[5] Amanda Bradbury, Eric Wiebe, Jessica Vandenberg, Jennifer Tsan, Collin Lynch, and Kristy Boyer. 2019. The Interface Design of a Collaborative Computer Science Learning Environment for Elementary Aged Students. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*. Sage publications Sage CA: Seattle, WA, 493–497.

[6] Brian Broll, Péter Völgyesi, János Sallai, and Akos Lédeczi. 2016. NetsBlox: A visual language and web-based environment for teaching distributed programming. (2016).

[7] Shannon Campe, Jill Denner, Emily Green, and David Torres. 2019. Pair programming in middle school: variations in interactions and behaviors. *Computer Science Education* 0, 0 (2019), 1–25. https://doi.org/10.1080/08993408.2019.1648119 arXiv:https://doi.org/10.1080/08993408.2019.1648119

[8] Alistair Cockburn. 2004. *Crystal clear: a human-powered methodology for small teams*. Pearson Education.

[9] Neil Davidson and Claire Howell Major. 2014. Boundary crossings: Cooperative learning, collaborative learning, and problem-based learning. *Journal on excellence in college teaching* 25 (2014).

[10] Jill Denner, Linda Werner, Shannon Campe, and Eloy Ortiz. 2014. Pair programming: Under what conditions is it advantageous for middle school students? *Journal of Research on Technology in Education* 46, 3 (2014), 277–296.

[11] Prasun Dewan, Puneet Agarwal, Gautam Shroff, and Rajesh Hegde. 2009. Distributed side-by-side programming. In *Proceedings of the 2009 ICSE workshop on cooperative and human aspects on software engineering*. IEEE Computer Society, 48–55.

[12] Alejandro Echeverría, Matías Améstica, Francisca Gil, Miguel Nussbaum, Enrique Barrios, and Sandra Leclerc. 2012. Exploring different technological platforms for supporting co-located collaborative games in the classroom. *Computers in Human Behavior* 28, 4 (2012), 1170–1177.

[13] Manuel Fernández, Rupert Wegerif, Neil Mercer, and Sylvia Rojas-Drummond. 2002. Re-conceptualizing "scaffolding" and the zone of proximal development in the context of symmetrical collaborative learning. *Journal of Classroom Interaction* 36, 2/1 (2002), 40–54.

[14] Arthur C Graesser, Cathy L McMahen, and Brenda K Johnson. 1994. Question asking and answering. (1994).

[15] Lesley G Hathorn and Albert L Ingram. 2002. Cooperation and collaboration using computer-mediated communication. *Journal of Educational Computing Research* 26, 3 (2002), 325–347.

[16] Cristián Infante, Pedro Hidalgo, Miguel Nussbaum, Rosa Alarcón, and Andrés Gottlieb. 2009. Multiple Mice based collaborative one-to-one learning. *Computers & Education* 53, 2 (2009), 393–401.

[17] Patrick J Leman. 2015. How do groups work? Age differences in performance and the social outcomes of peer collaboration. *Cognitive science* 39, 4 (2015), 804–820.

[18] Patrick J Leman and Gerard Duveen. 1996. Developmental differences in children's understanding of epistemic authority. *European Journal of Social Psychology* 26, 5 (1996), 683–702.

[19] Colleen M Lewis. 2011. Is pair programming more effective than other forms of collaboration for young students? *Computer Science Education* 21, 2 (2011), 105–134.

[20] Colleen M Lewis and Niral Shah. 2015. How Equity and Inequity Can Emerge in Pair Programming. In *Proceedings of the Eleventh annual International Conference on International Computing Education Research*. ACM, 41–50.

[21] Neil Mercer. 2002. *Words and minds: How we use language to think together*. Routledge.

[22] Jerzy R Nawrocki, Michał Jasiński, Łukasz Olek, and Barbara Lange. 2005. Pair programming vs. side-by-side programming. In *European Conference on Software Process Improvement*. Springer, 28–38.

[23] John T Nosek. 1998. The case for collaborative programming. *Commun. ACM* 41, 3 (1998), 105–108.

[24] Jean Piaget. 2013. *The construction of reality in the child*. Routledge.

[25] Lutz Prechelt, Ulrich Stärk, and Stephan Salinger. 2008. 7 types of cooperation episodes in Side-by-Side programming. (December 2008).

[26] Stacey D Scott, Regan L Mandryk, and Kori M Inkpen. 2003. Understanding children's collaborative interactions in shared environments. *Journal of Computer Assisted Learning* 19, 2 (2003), 220–228.

[27] N Shah, C Lewis, and R Caires. 2014. Analyzing equity in collaborative learning situations: A comparative case study in elementary computer science. In *Proceedings for the 11th International Conferences of the Learning Sciences (ICLS)*. 495–502.

[28] Jennifer Tsan, Collin F Lynch, and Kristy Elizabeth Boyer. 2018. "Alright, what do we need?": A study of young coders' collaborative dialogue. *International Journal of Child-Computer Interaction* (2018).

[29] Linda L Werner, Brian Hanks, and Charlie McDowell. 2004. Pair-programming helps female computer science students. *Journal on Educational Resources in Computing (JERIC)* 4, 1 (2004), 4.

[30] Laurie Williams, Robert R Kessler, Ward Cunningham, and Ron Jeffries. 2000. Strengthening the case for pair programming. *IEEE software* 17, 4 (2000), 19–25.

[31] Laurie Williams, Eric Wiebe, Kai Yang, Miriam Ferzli, and Carol Miller. 2002. In support of pair programming in the introductory computer science course. *Computer Science Education* 12, 3 (2002), 197–212.

[32] Zarifa Zakaria, Danielle Boulden, Jessica Vandenberg, Jennifer Tsan, Collin Lynch, Eric Wiebe, and Kristy Elizabeth Boyer. 2019. Collaborative Talk Across Two Pair-Programming Configurations. (2019), 224–231.