The Foundations of Collaborative Programming by Elementary-Aged Children

Jessica Vandenberg, Jennifer Tsan, Zarifa Zakaria, Collin Lynch, Kristy Elizabeth Boyer, and Eric Wiebe

Abstract Until recently, collaborative programming in elementary contexts largely involved traditional one-computer pair programming, more commonly used in industry and university settings. This chapter outlines the historical background of traditional pair programming and if and how its use with elementary-aged children holds as much promise. We then explore alternative paradigms, which generally include the addition of a second computer; we call this configuration two-computer pair programming. Harnessing the advances in programming applications that permit synchronized collaboration, we then report on our work with both one- and two-computer pair programming in block-based programming environments. We conclude by providing a summary of student experiences and opinions on the one- and two-computer studies, and a set of actionable guidelines for practitioners who would like to implement these different collaborative paradigms in their elementary computer science lessons and digitally mediated team learning (DMTL) researchers who will develop support features in these types of environments.

J. Vandenberg (✉) · Z. Zakaria
Educational Psychology, North Carolina State University, Raleigh, NC, USA
e-mail: jvanden2@ncsu.edu; zzakari@ncsu.edu

J. Tsan
Department of Computer Science, University of Chicago, Chicago, Illinois
e-mail: jennifer.tsan@uchicago.edu

C. Lynch
Computer Science, North Carolina State University, Raleigh, NC, USA
e-mail: cflynch@ncsu.edu

K. E. Boyer
Computer & Information Science & Engineering, University of Florida, Gainesville, FL, USA
e-mail: keboyer@ufl.edu

E. Wiebe
STEM Education, North Carolina State University, Raleigh, NC, USA
e-mail: eric_wiebe@ncsu.edu

L. O. Campbell et al. (eds.), *Perspectives on Digitally-Mediated Team Learning*, Educational Communications and Technology: Issues and Innovations,
https://doi.org/10.1007/978-3-030-77614-5_4

Introduction

Collaboration

At its foundation, *collaborative learning* is when two or more students undertake learning together (Dillenbourg, 1999). Researchers and practitioners have considerable latitude in defining collaborative learning; however, as group size may range from dyads to an entire class, the learning may be proximal or distal, and together may mean physically close to or asynchronously linked. In computer-supported collaborative learning (CSCL) contexts, Dillenbourg (2002) suggests that it is more appropriate for research on collaborative learning to focus specifically on how students interact as well as the affective and cognitive implications of these interactions. He maintains this work can be done by structuring the collaborative process to produce desired outcomes and/or by regulating student interactions. Thus, CSCL research necessarily aligns expertise and knowledge from education, educational technology, and computer science (Järvelä & Hadwin, 2013).

Successful collaboration comprises discussion, shared decision-making, and joint engagement (Roschelle & Teasley, 1995). One of the affordances of a CSCL environment is that these processes can be made more obvious to the participants, practitioners, and researchers alike (Shawky et al., 2014). Learners in a CSCL context co-construct their knowledge and manage their own and others' learning in ways that are often visible, and this can occur through the use of shared displays and discussion that makes learners' thinking conspicuous (Miyake, 1997). Within the discipline of computer science, one of the ways to facilitate collaborative co-construction of knowledge is by using a pair programming approach embedded in a CSCL activity. Traditionally, pair programming requires that two programmers work together on a shared activity on a single computer (Williams et al., 2000). However, with advances in functionality of technological applications that support collaboration and team-learning, new conceptualizations are emerging. Both traditional and newer conceptualizations of pair programming will be discussed below.

Understanding Collaborative Processes

Before discussing pair programming, it will be important to understand the foundational collaborative processes that pair programming is supposed to encourage and support. By encouraging students to problem-solve on a joint task, they must consider their different ideas and perspectives, varied strategies they might employ, how to overcome conflict, and the processes involved in collaborative decision-making. The study of computer-supported collaborative processes has resulted in a set of characteristics surmised to define the quality of collaboration, including communication, joint information processing, coordination, interpersonal relationships, and motivation (Meier et al., 2007). We use these five characteristics to explore how

different collaborative (pair) programming configurations will impact different facets of collaborative processes in different ways. We briefly outline each of them here.

Communication in a collaborative activity is operationalized as the establishment of joint ideas, expectations, and assumptions as well as the use of verbal and nonverbal cues that signal acknowledgement of ideas, attending to the task and speaker, and transitions involving turn-taking.

Joint information processing is the notion that collaborators have joint and individual knowledge but by incorporating individuals' knowledge, they are better equipped to leverage the resources of one another and to come to a more thoughtful, reasoned consensus.

Coordination is largely how collaborators organize themselves and their work; in particular, it is concerned with how they manage the task, their time, and technical aspects.

The **interpersonal relationship** between collaborators is most successful when balanced, or symmetrical; when socially fractious or otherwise imbalanced, the collaborators can engage in unproductive and off-task conflict.

Motivation refers to an individual's dedication to the task, including focused attention and engendering an effective learning environment.

One-Computer (1C) Pair Programming

Foundational 1C Research

Computer science education literature is replete with pair programming studies. In the classic implementation of this configuration, two programmers work collaboratively on one computer and operate using the roles of *driver* and *navigator* (Williams et al., 2000). The driver makes changes to the code and is in control of the mouse/trackpad and keyboard. The navigator checks the driver's work and looks ahead to scan for problems. Both programmers are expected to talk continuously about their work, collaboratively problem solve, and switch roles after a set amount of time or when a portion of the task has been completed. This pedagogical configuration has been used in industry (Beck, 1999; Canfora et al., 2007), in undergraduate classes (Williams et al., 2000), and in high school (Missiroli et al., 2016). We refer to this configuration as *one-computer pair programming* (1C) because both programmers share one computer.

Universities began to use 1C in their computer science classrooms soon after it was introduced as an accepted industry practice. At the university level, 1C studies have revealed that students had a higher success rate in their classes, performed better on midterms and tests and performed better on projects (Werner et al., 2004; Williams et al., 2002) than students that worked individually. 1C programmers also produced higher quality code than individual programmers (Williams & Upchurch, 2001). Additionally, Werner et al. (2004) found that 1C programming helped to

improve undergraduate students' confidence, especially female students, and thus could potentially help with female retention challenges. Although research has documented many benefits to 1C programming, there is evidence to suggest that when one collaborator starts out more knowledgeable than another, the more knowledgeable peer tends to take control, leaving the other partner confused, disengaged, or unhappy (Braught et al., 2010).

Research in K-12 contexts is a more recent contribution. Middle-school 1C programmers not only increased their Alice programming knowledge but did better in computational thinking post-tests than did individual programmers (Denner et al., 2014). Werner et al. (2013) found that when middle school friends are paired together, if one student is more confident than the other, then the more confident student's programming knowledge improves when working with a less confident partner with more prior knowledge. Recently, researchers have turned their attention to deep quantitative and qualitative analyses to understand the pair programming process and the programmers' affective and cognitive states (Celepkolu & Boyer, 2018; Rodríguez et al., 2017; Toma & Vahrenhold, 2018).

Newer Interest in 1C with Younger Students

Newer work on elementary students' pair programming has often focused on their discourse and collaborative processes. In an early study of an interdisciplinary activity with younger students using programming and knowledge of science topics, such as the human brain, Kafai and Ching (2001) found that when upper elementary students were planning and designing a science website, they engaged in discussions of scientific concepts more when designing screens. They also found that students who had more experience in software design contributed more to discussions about the scientific topics within the context of design.

Other researchers that have taken a closer look at younger students' collaborative processes have found that, on average, pairs of Latino/a students used more nonverbal behaviors than pairs of Caucasian students (Ruvalcaba et al., 2016). The equity of a collaborative relationship depends in part on how the individuals' goals align (Lewis & Shah, 2015) and the way the students position each other socially (Shah et al., 2014). Elementary students often engaged in both problem-specific discussions (Baytak & Land, 2011; Israel et al., 2017) and discussions of their achievements, as well as more general off-topic conversations (Israel et al., 2017). Problem-specific questions involved talking through the problem (Israel et al., 2017), asking for help (Baytak & Land, 2011), and exchanging ideas (Baytak & Land, 2011). In a study with younger children, Fessakis et al. (2013) found that the students' class interactions fell into one of five categories: competition, interference concerning command proposals and instructions, collaboration, moral support, or dialogue development among the rest of the children.

Challenges to Using 1C

Lewis and Shah (2015) and Shah et al. (2014) analyzed the frequency of the students' communication with one another as well as the content of their discussion. The authors investigated four pairs of students with one student, Jason, as a student in all pairs. Two of his collaborative relationships were more equitable than the other two. As a result, the authors reached the conclusion that equity in a collaborative relationship may be contextualized based on the lessons students are working on, and some students' desire to complete projects quickly may lead to inequitable relationships. Their curriculum was self-paced and may have contributed to students' focus on completing the projects quickly.

In a study of equity in 1C, researchers analyzed how a pair of girls in a high-school elective on the topic of digital making positioned themselves via speech and computer usage (Deitrick et al., 2016). The analysis leveraged positioning theory (Van Langenhove & Harré, 1999), in which the roles of agents (students) interacting with one another are considered fluid and are changed according to the interaction. The study of girls' collaboration found that one student established herself in a more knowledgeable and authoritative position by speaking more, giving commands, and maintaining control of the equipment (Deitrick et al., 2016).

In brief, these studies indicate that traditional 1C can be problematic because it can lead to inequitable relationships and conflict. Tsan et al. (2019) found that upper elementary students in 1C activities often displayed a series of conflicts, which mostly revolved around implementation of ideas, turn-taking, and other equipment issues. However, these noted challenges open up opportunities for a reevaluation of using 1C to explore alternative paradigms.

Collaborative Processes in 1C

Meier et al. (2007) framework can be used to summarize how both technology and student age can shape the patterns of collaboration in a 1C pair programming environment.

Communication

Pair programming in a 1C configuration requires that both programmers communicate continuously and effectively. When executed appropriately, both programmers employ varied strategies to acknowledge each other's contributions, such as verbal utterances, establishing eye contact, using gestures, such as head nodding, or following each other's suggestions. Levels of communication may differ by age. In fact, Tsan et al. (2018) found that most elementary 1C programmers do not verbally acknowledge their partner's contribution.

Joint Information Processing

The sharing of information is essential in any collaborative task. In 1C, information sharing may be complicated by the fact that only one programmer is in control of the input devices and therefore seemingly in charge of final coding decisions. Moreover, resolving conflicts or coming to consensus on next steps is challenging for young students who do not often have the social skills and experience to settle disagreements on their own.

Coordination

How students plan and manage their time and task components is important for any pair programming activity. In a 1C configuration, coordination may look like outlining which tasks need to occur first and which programmer/driver will accomplish what. With timing, external controls (e.g., a clock) are often used to enforce switch points.

Interpersonal Relationship

A 1C configuration has the potential to exacerbate an imbalance in a pair's relationship as many young programmers prefer the driver role to the navigator role, often resulting in diminished interactions on the navigator's side. Control of the preferred driver's role can thus be utilized as an exercise in power.

Motivation

Maintaining motivation toward any activity is important for programmers to elicit learning gains. In 1C, maintaining motivation may be problematic for the navigator as young programmers struggle with embracing the importance of this role and effectively adhering to its purposes and expectations.

Alternative Paradigms

Other researchers have expanded the conceptualization of what constitutes pair programming by augmenting the number of computers being used. In a sixth-grade coding camp, Lewis (2011) compared traditional pair programming with "intermittent collaboration," a condition in which paired students worked on their own computers but were required to sit next to one another, to discuss their work and problems every 5 min, and to assist each other before asking the teacher for help. The *intermittent collaborators* completed their work more quickly and viewed Scratch and

programming more favorably than the traditional pair programmers. This collaborative programming configuration is not new (e.g., Cockburn, 2004; Nawrocki et al., 2005; Prechelt et al., 2008). We term this new paradigm as *two-computer (2C) pair programming* (Tsan et al., 2020). *2C pair programming,* or 2C for short*,* occurs when two programmers each have 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 (Fig. 1).

Qualitative analyses of 2C programmers indicate that, although working independently may be the primary mode used because there are two sets of unlinked input devices, pairs engaged one another in conversation for very specific purposes: to combine their work, to share information, to debug the code, or to talk about work strategies or next steps (Prechelt et al., 2008). This fluid and purposeful conversation is possible because of their physical proximity. Cockburn (2004) supported this physical separation of 2C programmers, noting that traditional pair programming left drivers feeling watched by the navigators and with minimal time to work on other tasks. In a study with upper level undergraduates, Nawrocki et al. (2005) compared traditional pair programmers to 2C programmers on completion time/effort, familiarity with code, and the programmers' impressions with their programming condition. Seventy percent of programmers said they preferred 2C programming.

Dewan et al. (2009) put further separation between programmers by introducing the idea of distributed 2C programming. Here, programmers worked remotely but each had a second *awareness computer* that displayed their partner's real-time progress. The authors describe distributed 2C programming as a superset of solo and traditional pair programming in that participants can choose to program individually, as a driver or navigator, or alongside their partner.

A recent contribution to the field is *distributed computing*. A programming environment such as NetsBlox (Broll et al., 2016) permits this type of collaborative work. It is a visual block-based programming platform designed to teach younger students and other novices distributed collaborative computing. Students can invite collaborators—other learners from within the same classroom to others around the world—to help create content, and the code updates on one screen appear in rapid succession on the other. This type of synchronous distributed computing removes

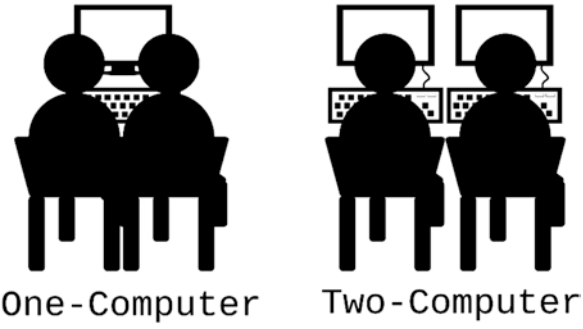

Fig. 1 One-computer and two-computer configurations

the need for "awareness computers" and permits students to collaborate wherever there is an internet connection.

Two recent studies that use NetsBlox in a distributed 2C configuration with upper elementary students show promise in structuring young programmers' experiences with block-based applications. Zakaria et al. (2019) found that this configuration could lead to cooperative work, rather than collaborative work. Cooperative work is that which occurs in tandem and is typified by students dividing tasks to be completed individually, whereas collaborative work is premised on the interdependence of students who complete work together (Hathorn & Ingram, 2002). Bradbury et al. (2019) summarized multiple studies and revealed that upper elementary students largely preferred the 2C configuration and that students perceived they learned more and had more hands-on experiences in this configuration.

Lytle et al. (2020) investigated three collaboration modes with middle- and high-school students' distributed 2C experiences with NetsBlox. The authors designed and compared three conditions: Pair-Separate, Pair-Together, and Pair-Puzzle. In Pair-Separate, the students work in the same project file but have separate tasks and code to complete. In Pair-Together, the students work in the same project file but have access to all code and have editing rights to their partner's script. In Pair-Puzzle, the students work similarly to Pair-Together although each student has a specific set of code blocks necessary to complete the task. Results indicate that students overwhelmingly preferred the Pair-Puzzle condition as the mutually dependent nature of this mode prompted students to talk with regularity and know that their limited blocks would result in the correct code.

Although not in NetsBlox, Deng (2017) further investigated how different collaboration models affect the collaboration process with undergraduate students. Deng implemented three collaboration models in MIT AppInventor, a block-based programming environment where users can create Android apps. The collaboration models were project-level collaboration (only one user can edit the code at a time), component-level collaboration (only one user can edit each component or set of blocks at a time), and real-time collaboration (any user can edit anything at any time). The author found that the real-time collaboration model yielded shorter turns and the lowest mistake rate. The component-level collaboration model yielded the lowest communication level and highest mistake rate. The project-level collaboration model yielded longer turns and the highest communication rate. Lytle et al.'s (2020) Pair-Separate and Pair-Together are most similar to Deng's (2017) component-level collaboration and real-time collaboration models, respectively. These collaboration models can also be implemented in NetsBlox for further research on how they affect students' collaboration processes.

Linked and Unlinked 2C

In our research, we have employed a number of collaboration coding taxonomies and analytic techniques to explore the interaction of configuration and pedagogy and how they might optimize Meier's categories. The review of CS education literature outlined above provided examples and inspiration for 2C programming configurations that could be explored in our own work.

For our work, we have two versions of 2C: *linked* and *unlinked*. For *Linked 2C*, the students work in a synchronized, shared development environment in the same project (like in Google docs). We investigated linked 2C programming by providing each child with a computer and synchronizing their workspaces over the web using the NetsBlox programming environment (Broll et al., 2016). For *Unlinked 2C*, the students do not have a shared development environment and each of them have a different copy of the project. This structure leads to dynamics that are similar to a 1C configuration with students communicating about their work, offering suggestions, and exchanging control of code, even though the driver/navigator roles are not strictly enforced.

Collaborative Processes in 2C

Most of the older studies noted previously emphasize adult students—with a few focused on intermediate programmers who were CS graduate students—and this work was only done with text-based languages. The more recent shift toward exploring how young students might benefit from a 2C programming context and use of block-based programming environments indicates elementary students may be more satisfied with 2C programming; however, they often work cooperatively rather than collaboratively, and more research is needed on the best way to leverage 2C programming and how we can build adaptive support features to help them. Here, we outline how Meier's framework can summarize elementary students' collaborative processes using 2C programming.

Communication

Pair programming in a 2C configuration requires effective communication between partners because the two programmers, with their individual workspaces, can work on different aspects of the programming activity. Therefore, they must verbally apprise each other of their thinking and actions in order to not impede individual and group progress. In an unlinked 2C configuration, this communication may be all the more vital as the two programmers do not have visual awareness of each other's coding actions.

Joint Information Processing

In 2C, sharing of information could be reduced if both programmers are able to edit the code. Students may exchange ideas less frequently during 2C and a teacher or system intervention may be necessary to help students have successful 2C interactions. In an unlinked 2C configuration, consensus may be less critical as the two programmers have the ability to make individual decisions. In a linked 2C configuration, consensus is necessary for both programmers to feel that their contributions to the common code are valued.

Coordination

Coordination is essential for the success of 2C implementation because the increase in input devices likely enhances the need for systematic actions. With the linked 2C configuration, students will need to decide who is editing which parts and when. Otherwise, they will likely encounter editing conflicts in the code and become confused about the output (Bradbury et al., 2019). In this linked environment, students may resort to a quasi-cooperative mode where they work in parallel on separate parts of code. In an unlinked 2C configuration, pre-task planning may help programmers determine how to chunk tasks by time. Additionally, if they are creating the exact same project in both workspaces, it may help ensure that they are both making the same edits in their workspaces.

Interpersonal Relationship

There is potential for programmers to maintain a more symmetrical interpersonal relationship balance in 2C because they can express individual autonomy through their individual workspace. This symmetry may vary by linked and unlinked configuration; however, in unlinked, the need to maintain a symmetrical relationship, although desired, is less imperative because the programmers can complete individual tasks.

Motivation

Maintaining motivation in a 2C configuration may be markedly different from that which occurs in 1C. The value students ascribe to their role and the contribution this role makes can shape motivation. In 2C, programmers have the opportunity to directly construct and edit code if they so choose. With linked 2C, students also have the option of assuming the navigator role as done in 1C and, for that reason, there is still the potential for one student to move into a more passive role, disengaging while the other student completes the tasks at hand. In that way, it is still

important to maintain motivation for students using linked 2C as there is for students using 1C.

Current Work: 2C

Study Contexts

In the work done by our team described in this section, the students experienced a curriculum that covered concepts such as algorithms, conditionals, loops, broadcasting, and user input. As part of this curriculum, the students were taught about both 1C and linked 2C programming. They were introduced to the roles and responsibilities of the driver and navigator in 1C, and they were taught about the importance of talking through their decision-making process in both configurations. The teachers shared digital posters for both 1C and 2C (modified versions presented in Fig. 2) or had printed versions of the posters that were placed in front of the students and reminded students of how they should work in each configuration. Our studies took place in two suburban fifth-grade classrooms at Clark and Frederick Elementary (pseudonyms). All students experienced 1C and 2C. We alternated the programming paradigm for every lesson.

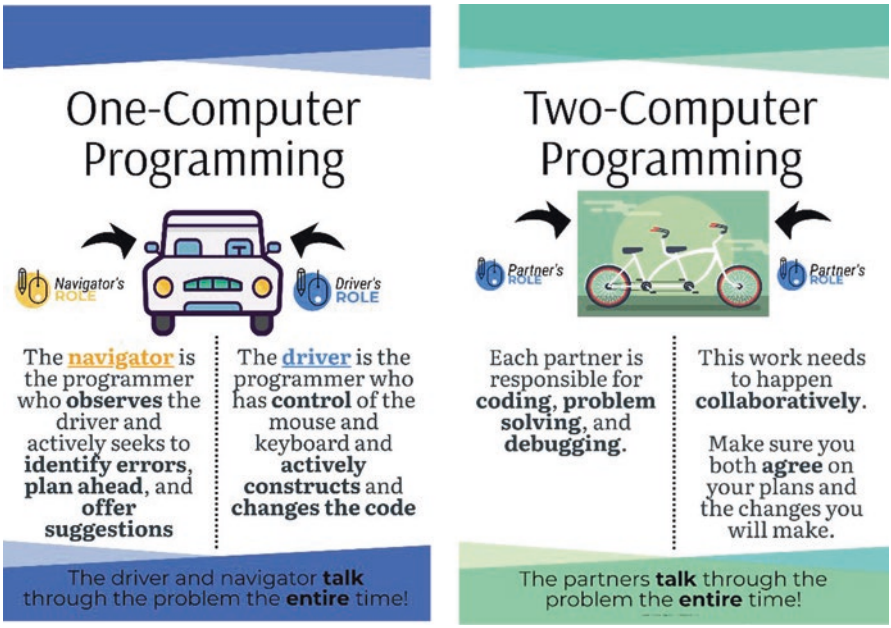

Fig. 2 Posters of one-computer and two-computer pair programming instructions

Collaborative Regulation of Learning

At Clark Elementary, 68 fifth-grade students participated in the study (29 girls, 39 boys). The students, paired according to the teacher's perceptions of the students' prior collaborative behaviors, were taught the coding lessons by the school's media center teacher during the students' dedicated weekly media special. Students used both 1C and 2C programming and were video recorded each time they programmed. Videos were transcribed verbatim and annotated using a multidimensional coding scheme based on collaborative regulation of learning (Janssen et al., 2012) and how students' discourse indicated they were working together (Kumpulainen & Mutanen, 1999). Table 1 is a modified version of this framework.

Over the course of the study, there were noticeable differences by 1C and 2C configuration in how students spoke to each other. Overall, on 1C days, students uttered more *Monitoring*, *Agreement*, *Collaboration*, and *Confusion* statements than on 2C days, whereas on 2C days students uttered more statements tagged as *Planning* and *Evaluation*. However, we found that students' discourse was drastically different from the initial 2 days of the study compared to the final 2 days; regardless of configuration, on the final days, students uttered more *Confusion*, *Disagreement*, and *Individualistic* statements compared to the first 2 days.

Table 1 Modified collaborative regulation of learning framework (based on Janssen et al., 2012; Kumpulainen & Mutanen, 1999)

Dimension	Code	Definition	Examples
Task regulation	Planning	Discussion of the task, how to complete it, deciding which strategies to employ, responsibilities students will take on	Let us start by picking a background
	Monitoring	Discussion of performance and progress, specific mention of strategies being used to approach the task, mentions of time	The glide block would work better last time, so let us try that
	Evaluation	Review of performance and progress, includes appraisals of task difficulty	That was harder than I thought it would be
Social	Collaborative	Actively engaging with partner, attempts to maintain symmetrical contributions	Let us change it so she says "hello" for longer, don't you think?
	Agreement	Acknowledgements and affirmations, most often in response to a partner's contribution	Oh yeah! Yes
	Tutoring	Asking for or offering help/assistance	Hey, how do I add another sprite?
	Disagreement	Social or academic conflict	I will delete it if you write that in there
	Confusion	Failure to understand the partner or the task, often accompanied by a question	That is not what I was thinking
	Individualistic	Working independently with no clear attempt to involve the partner	(These examples often looked like self-talk in proximity to another)

When we consider Meier's collaborative processes as demonstrated here, we found that there was a unique interaction between configuration and time. In the first 2 days, students communicated effectively, largely coordinated their efforts well, maintained their **motivation** and an appropriately balanced **interpersonal relationship**, and successfully engaged in **joint information processing**. However, on the final days, the students, not having developed a sense of competence in either the programming environment or configuration, struggled, leading to confusion, disagreement, and more individual work. Their **interpersonal relationships** were unbalanced and contentious, leading to ineffective **communication** and a lack of **joint information processing**.

Investigating Types of Talk

At Frederick Elementary, we worked with a gifted classroom comprising a total of 11 students who were taught by one of the authors about both 1C and linked 2C programming. We utilized a coding scheme aligned with Mercer's original IDZ (2000) framing guided by T'sas' (2018) articulation of Mercer's three talk types (Exploratory, Cumulative, and Disputational) to code student transcripts (Table 2).

Using time interval coding (Bakeman, 2000), we found that, overall, students used Cumulative conversation more than the other two categories in both 1C and 2C configurations (Fig. 3). 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 collaborative relationships devolving into cooperative relationships,

Table 2 Combination framework for types of talk (based on Mercer, 2000; T'sas, 2018)

	Exploratory	Cumulative	Disputational
Major characteristics	Challenge, alternative hypothesis, critical reasoning	Uncritical addition of ideas, agreement	Disagreement without critical reasoning
Elaborated characteristics	● Offered alternative hypothesis ● Initiations challenged, and counter-challenged followed by consensus ● Justifications given ● Joint acceptance	● Agreement without critical discussion ● Friendly and conflict avoidance ● Positive but uncritical adding of ideas ● Superficial amendments	● Disagreement without outcome ● Individualized decision-making ● Initiations directly rejected ● No/little constructive criticism ● Counter proposition with no consensus ● No resolutions

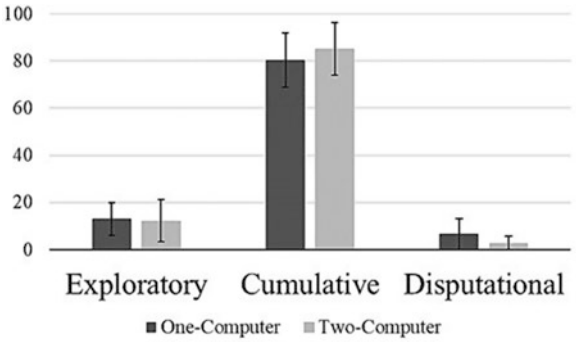

Fig. 3 Percentage of each type of talk per minute, by configuration

where students worked parallel rather than focused on the same immediate task (Davidson & Major, 2014). However, partners still consulted each other regularly and had a balance of opportunities to challenge and explain their thoughts to each other in 2C.

In this work, we focused on students' **communication** with a focus on the types of talk that were used by the students. Although there were not any differences with types of talk, there were differences in patterns of talk between 1C and 2C pairs. There was also a difference in how they **coordinated**, how much they **communicated**, and whether they used **joint information processing**.

Unlinked 2C

The prior studies indicate that a linked configuration helped shape students' collaboration. As a point of contrast, we now report on an unlinked 2C configuration study. We selected six dyads, three from each condition, based on the video and audio clarity. Dyads created identical programs on each of their computers. Students in the feedback condition received structured feedback on key characteristics of Exploratory talk (i.e., *challenging partners with questions, sharing alternative ideas, justifying ideas, or disagreements*).

To better understand how Mercer's (2000) three types of talk are formed in a pair programming context, we needed more information about conversational categories that would combine into these types of talk than the aggregated scheme used in "Investigating Types of Talk" had provided. This abbreviated coding scheme is presented in Table 3.

The findings suggested that, overall, collaborative talk was significantly higher in dyads who received feedback and instruction. The most important indication of this study is that nearly all of the Exploratory categories (*alternative idea, justification, disagreement followed by justification*) were used at significantly higher rates in the experimental condition, except for *higher order questions*. Instead of *higher order questions*, use of *simple questions*—questions that do not challenge partner's

Table 3 Abbreviated framework for types of talk

	Categories	Description	Examples
Cumulative	*Simple question*	Question about a process or fact (any type of questions which are not higher order)	"Which one?" "What are you doing?"
Exploratory	*Higher order question*	Questions that challenge their partner's ideas. These questions should be asked for reasoning	"Why did you move it?" "How do you know?"
	Justification	Student justifies their idea, change, or step with reasons	"Five seconds is too long"

ideas—was found to be used significantly more by the experimental condition. Reflecting on the content of the feedback, we believe that our prompts may need clarification on how those questions are different. The findings lead us to examine an intervention that can be utilized to engage students in higher levels of collaboration as exemplified in Mercer's Exploratory talk.

With the 2C unlinked pairs, the students were required to **coordinate** more than the students that used 2C linked pair programming in the previous study. Both students in each pair needed to maintain **motivation** in order to complete the project in their workspaces. Comparing the feedback and control conditions, we found that the **communication** was better with the pairs in the feedback condition. Their communication consisted of a larger percentage of Exploratory talk and *simple questions*.

Takeaways and Conclusions

Summary of Research Findings

Benefits

1C	2C linked	2C unlinked
● More occurrences of exploratory conversation before and after disputational (**communication**) ● Students monitored their work during the task, collaborated and agreed (**communication, joint information processing**)	● Students engaged in more planning (**coordination**) and evaluation (**joint information processing**) actions prior to and after the task ● More hands-on experience than 1C (**motivation**)	● Students were able to each work on their own computer and code (**motivation**)

Challenges

1C	2C linked	2C unlinked
● Students expressed more confusion (**communication, joint information processing**)	● Students often resorted to cooperation, working in parallel, instead of collaboration (**coordination**)	● Context demanded students coordinate through self-explanation to create the same project on multiple devices (**coordination**)
• Waiting to be in the driver role (**motivation, interpersonal relationship**)	● Context demanded students coordinate to prevent coding conflicts (**coordination**)	● Difficulty seeing each other's work (**joint information processing)**
	● Difficulty seeing each other's work (**joint information processing)**	

Summary of Student Experiences and Opinions

To better understand how students perceived their work in 1C and 2C, we held focus groups with students at one of the study sites. The majority of students (12 out of 15) preferred 2C over 1C (Bradbury et al., 2019).

Students reported the following challenges in 1C. One student tried to retain the driver role, which the students preferred to navigating as they did not like waiting to work on the computer (**motivation, interpersonal relationship**). Moreover, students stated that they argued more often because their partner did not listen to them (**communication**, **interpersonal**). Lastly, students felt there was not enough hands-on experience in this configuration (**motivation**), which they clearly valued. In 2C, students largely reported technical difficulties with the two computers not synching quickly which led to a lag in one student seeing what their partner completed (**coordination**, **joint information processing**). Also, students complained of lack of space given that each student had their own computer.

Regarding the benefits of 1C, students reported that they learned more because they could see and help resolve their partner's mistakes (**communication**, **coordination**, **joint information processing**), and that they felt less cramped with only one computer. In 2C, students enjoyed that they each had a computer, leading to more hands-on experience (**motivation**), and that they learned more in this configuration as a result. Additionally, some reported that they cooperatively worked by breaking tasks into smaller subtasks (**communication**, **coordination**).

Set of Actionable Guidelines for Practitioners

Collaborative programming holds promise for advancing programmer confidence, satisfaction, knowledge, and enjoyment (e.g., Maguire et al., 2014; Rodríguez et al., 2017; Tsan et al., 2020). Traditional 1C programming may be better suited for older programmers, whose cognitive and social development better supports turn-taking, verbal scaffolding of a partner, conflict resolution, and joint problem-solving. 2C, either linked or unlinked, may support younger programmers, as such configurations tend to ease concerns about inequity, conflicts over turn-taking, and code implementation, and provide a more direct hands-on experience. Practitioners must weigh the logistical considerations of their classrooms (e.g., seating arrangements, number of devices, bandwidth strength, size of devices) alongside the learning needs and abilities of their students.

Campe et al. (2019) provide a toolkit for practitioners to use in K-12 classrooms when implementing traditional 1C programming. Their toolkit is research-based, gives practitioners suggestions on ways to pair students, and offers activities for enhancing collaboration and communication.

From our own research with upper elementary students, we offer the following final thoughts for practitioners.

- To allow for more student agency, 2C is likely a preferred option. In this configuration, students are encouraged to express their ideas while also negotiating toward a group outcome.
- When practitioners have students who would benefit from hands-on experience, 2C is the preferred option; however, when students would benefit from peer-to-peer learning, 1C may be best.
- Regardless of the configuration, practitioners likely need to support students' communication practices, in particular, with turn-taking and role-playing in 1C and with coordinating individual to group efforts in 2C.

Additionally, we suggest that researchers consider implementing the following software features for 1C and 2C programming.

- Icons of the students who are connected and a way for students to click on the icon to determine which sprite/background their partner(s) are working on.
- Showing the pointers in order to allow the students to gesture on the screen.
- Communication support in the interface. These supports could be in the form of virtual agents or prompts and collaboration scripts to reinforce effective communication between students; in particular, supports are needed to reinforce appropriate turn-taking in 1C, and verbal scaffolding of a partner, conflict resolution, and joint problem-solving strategies in both 1C and 2C.

References

Bakeman, R. (2000). Behavioral observation and coding. In H. T. Reis & C. M. Judd (Eds.), *Handbook of research methods in social and personality psychology* (pp. 138–159). Cambridge University Press.

Baytak, A., & Land, S. M. (2011). An investigation of the artifacts and process of constructing computers games about environmental science in a fifth-grade classroom. *Educational Technology Research and Development, 59*(6), 765–782. https://doi.org/10.1007/s11423-010-9184-z

Beck, K. (1999). Embracing change with extreme programming. *Computer, 32*(10), 70–77. https://doi.org/10.1109/2.796139

Bradbury, A., Wiebe, E., Vandenberg, J., Tsan, J., Lynch, C., & Boyer, K. (2019). The interface design of a collaborative computer science learning environment for elementary aged students. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting, 63*(1), 493–497. https://doi.org/10.1177/1071181319631155

Braught, G., MacCormick, J., & Wahls, T. (2010, March). The benefits of pairing by ability. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (pp. 249–253). https://doi.org/10.1145/1734263.1734348

Broll, B., Völgyesi, P., Sallai, J., & Lédeczi, A. (2016). *NetsBlox: A visual language and web-based environment for teaching distributed programming*. [White paper]. Vanderbilt University. https://netsblox.org/NetsBloxWhitePaper.pdf

Campe, S., Green, E., & Denner, J. (2019). *K-12 pair programming toolkit*. ETR.

Canfora, G., Cimitile, A., Garcia, F., Piattini, M., & Visaggio, C. A. (2007). Evaluating performances of pair designing in industry. *Journal of Systems and Software, 80*(8), 1317–1327. https://doi.org/10.1016/j.jss.2006.11.004

Celepkolu, M., & Boyer, K. E. (2018, February). Thematic analysis of students' reflections on pair programming in CS1. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (pp. 771–776).

Cockburn, A. (2004). *Crystal clear: A human-powered methodology for small teams*. Pearson Education.

Davidson, N., & Major, C. H. (2014). Boundary crossings: Cooperative learning, collaborative learning, and problem-based learning. *Journal on Excellence in College Teaching, 25*(3/4), 7–55.

Deitrick, E., Shapiro, R. B., & Gravel, B. (2016). How do we assess equity in programming pairs? In C. K. Looi, J. L. Polman, U. Cress, & P. Reimann (Eds.), *Transforming Learning, Empowering Learners: The International Conference of the Learning Sciences (ICLS) 2016, Volume 1*. International Society of the Learning Sciences.

Deng, X. (2017). *Group collaboration with app inventor*. (Doctoral dissertation, Massachusetts Institute of Technology).

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*(3), 277–296. https://doi.org/10.1080/15391523.2014.888272

Dewan, P., Agarwal, P., Shroff, G., & Hegde, R. (2009, May). Distributed side-by-side programming. In *2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering* (pp. 48–55). IEEE. https://doi.org/10.1109/CHASE.2009.5071410

Dillenbourg, P. (1999). Introduction: What do you mean by "collaborative learning"? In P. Dillenbourg (Ed.), *Collaborative learning: Cognitive and computational approaches* (pp. 1–19). Elsevier Science.

Dillenbourg, P. (2002). Over-scripting CSCL: The risks of blending collaborative learning with instructional design. In P. A. Kirschner (Ed.), *Three worlds of CSCL. Can we support CSCL* (pp. 61–91). Open Universiteit Nederland.

Fessakis, G., Gouli, E., & Mavroudi, E. (2013). Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study. *Computers & Education, 63*, 87–97. https://doi.org/10.1016/j.compedu.2012.11.016

Hathorn, L. G., & Ingram, A. L. (2002). Cooperation and collaboration using computer-mediated communication. *Journal of Educational Computing Research, 26*(3), 325–347. https://doi.org/10.2190/7MKH-QVVN-G4CQ-XRDU

Israel, M., Wherfel, Q. M., Shehab, S., Melvin, O., & Lash, T. (2017, August). Describing elementary students' interactions in K-5 puzzle-based computer science environments using the collaborative computing observation instrument (C-COI). In *Proceedings of the 2017 ACM Conference on International Computing Education Research* (pp. 110–117). https://doi.org/10.1145/3105726.3106167

Janssen, J., Erkens, G., Kirschner, P. A., & Kanselaar, G. (2012). Task-related and social regulation during online collaborative learning. *Metacognition and Learning, 7*(1), 25–43. https://doi.org/10.1007/s11409-010-9061-5

Järvelä, S., & Hadwin, A. F. (2013). New frontiers: Regulating learning in CSCL. *Educational Psychologist, 48*(1), 25–39. https://doi.org/10.1080/00461520.2012.748006

Kafai, Y. B., & Ching, C. C. (2001). Affordances of collaborative software design planningfor elementary students' science talk. *The Journal of the Learning Sciences, 10*(3), 323–363. https://doi.org/10.1207/S15327809JLS1003_4

Kumpulainen, K., & Mutanen, M. (1999). The situated dynamics of peer group interaction: An introduction to an analytic framework. *Learning and Instruction, 9*(5), 449–473. https://doi.org/10.1016/S0959-4752(98)00038-3

Lewis, C. M. (2011). Is pair programming more effective than other forms of collaboration for young students? *Computer Science Education, 21*(2), 105–134. https://doi.org/10.1080/08993408.2011.579805

Lewis, C. M., & Shah, N. (2015, August). How equity and inequity can emerge in pair programming. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (pp. 41–50). https://doi.org/10.1145/2787622.2787716

Lytle, N., Milliken, A., Cateté, V., & Barnes, T. (2020, February). Investigating different assignment designs to promote collaboration in block-based environments. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (pp. 832–838). https://doi.org/10.1145/3328778.3366943

Maguire, P., Maguire, R., Hyland, P., & Marshall, P. (2014). Enhancing collaborative learning using paired-programming: Who benefits? *AISHE-J: The All Ireland Journal of Teaching and Learning in Higher Education, 6*(2), 1411–1425.

Meier, A., Spada, H., & Rummel, N. (2007). A rating scheme for assessing the quality of computer-supported collaboration processes. *International Journal of Computer-Supported Collaborative Learning, 2*(1), 63–86. https://doi.org/10.1007/s11412-006-9005-x

Mercer, N. (2000). *Words and minds: How we use language to think together*. Routledge.

Missiroli, M., Russo, D., & Ciancarini, P. (2016, May). Learning agile software development in high school: An investigation. In *Proceedings of the 38th International Conference on Software Engineering Companion* (pp. 293–302). https://doi.org/10.1145/2889160.2889180

Miyake, N. (1997). Making internal processes external for constructive collaboration. In *Proceedings Second International Conference on Cognitive Technology Humanizing the Information Age* (pp. 119–123). IEEE Computer Society. https://doi.org/10.1109/CT.1997.617690

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. https://doi.org/10.1007/11586012_4

Prechelt, L., Stärk, U., & Salinger, S. (2008). *7 types of cooperation episodes in side-by-side programming* (Technical Report B-08-17). Freie Universität Berlin, Institut für Informatik.

Rodríguez, F. J., Price, K. M., & Boyer, K. E. (2017, March). Exploring the pair programming process: Characteristics of effective collaboration. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 507–512). https://doi.org/10.1145/3017680.3017748

Roschelle, J., & Teasley, S. D. (1995). The construction of shared knowledge in collaborative problem solving. In *Computer Supported Collaborative Learning* (pp. 69–97). Springer. https://doi.org/10.1007/978-3-642-85098-1_5

Ruvalcaba, O., Werner, L., & Denner, J. (2016, February). Observations of pair programming: Variations in collaboration across demographic groups. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 90–95). https://doi.org/10.1145/2839509.2844558

Shah, N., Lewis, C., & Caires, R. (2014). Analyzing equity in collaborative learning situations: A comparative case study in elementary computer science. In *Proceedings of the International Conference of the Learning Sciences, Colorado* (pp. 495–502). International Society of the Learning Sciences. https://doi.org/10.22318/icls2014.495

Shawky, D., Badawi, A., Said, T., & Hozayin, R. (2014, December). Affordances of computer-supported collaborative learning platforms: A systematic review. In *2014 International Conference on Interactive Collaborative Learning (ICL)* (pp. 633–651). https://doi.org/10.1109/ICL.2014.7017846

Toma, L., & Vahrenhold, J. (2018, August). Self-efficacy, cognitive load, and emotional reactions in collaborative algorithms labs: A case study. In *Proceedings of the 2018 ACM Conference on International Computing Education Research* (pp. 1–10). https://doi.org/10.1145/3230977.3230980

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). https://doi.org/10.1145/3159450.3159507

Tsan, J., Vandenberg, J., Fu, X., Wilkinson, J., Boulden, D., Boyer, K. E., Lynch, C. & Wiebe, E. (2019, February). An investigation of conflicts between upper-elementary pair programmers. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 1264). https://doi.org/10.1145/3287324.3293799

Tsan, J., Vandenberg, J., Zakaria, Z., Wiggins, J. B., Webber, A. R., Bradbury, A., Lynch, C., Wiebe, E. & Boyer, K. E. (2020, February). A comparison of two pair programming configurations for upper elementary students. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (pp. 346–352). https://doi.org/10.1145/3328778.3366941

T'sas, J. (2018). *Learning outcomes of exploratory talk in collaborative activities*. (Doctoral dissertation, University of Antwerp).

Van Langenhove, L., & Harré, R. (1999). Introducing positioning theory. In *Positioning theory: Moral contexts of intentional action* (pp. 14–31). Blackwell.

Werner, L., Denner, J., Campe, S., Ortiz, E., DeLay, D., Hartl, A. C., & Laursen, B. (2013, March). Pair programming for middle school students: Does friendship influence academic outcomes? In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (pp. 421–426). https://doi.org/10.1145/2445196.2445322

Werner, L. L., Hanks, B., & McDowell, C. (2004). Pair-programming helps female computer science students. *Journal on Educational Resources in Computing (JERIC), 4*(1), 4-es. https://doi.org/10.1145/1060071.1060075

Williams, L., Kessler, R. R., Cunningham, W., & Jeffries, R. (2000). Strengthening the case for pair programming. *IEEE Software, 17*(4), 19–25. https://doi.org/10.1109/52.854064

Williams, L., & Upchurch, R. L. (2001). In support of student pair-programming. *ACM SIGCSE Bulletin, 33*(1), 327–331. https://doi.org/10.1145/366413.364614

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*(3), 197–212. https://doi.org/10.1076/csed.12.3.197.8618

Zakaria, Z., Boulden, D., Vandenberg, J., Tsan, J., Lynch, C., Wiebe, E., & Boyer, K. (2019, June). Collaborative talk across two pair-programming configurations. In *International Conference on Computer-Supported Collaborative Learning (CSCL)* (pp. 224–231).