PearProgram: A More Fruitful Approach to Pair Programming

Maxwell Bigman
Stanford University
mbigman@stanford.edu

Ethan Roy
Stanford University
ethanroy@stanford.edu

Jorge Garcia
Stanford University
jorgeedu@stanford.edu

Miroslav Suzara
Stanford University
msuzara@stanford.edu

Kaili Wang
Stanford University
kkwang22@stanford.edu

Chris Piech
Stanford University
piech@cs.stanford.edu

ABSTRACT

In this paper we present *PearProgram*, a hybrid learning and research tool that helps introductory Computer Science (CS) students learn how to pair program, including in remote learning environments. Grounded in theory from the Learning Sciences, the tool – a collaborative, online IDE – has two primary goals: 1) to help introductory CS students achieve pair programming success; and 2) to research what factors contribute to pairs that have beneficial outcomes. We present our learnings from the use of *PearProgram* in three remote introductory CS courses: a CS1 course, and two large international courses, including one for high school students. Teacher and student users responded positively to *PearProgram*, and use of the tool was associated with beneficial learning outcomes in these online learning environments. Our research opens many future research directions for (remote) pair programming, and indicates practices that may prove useful for CS educators at all levels.

CCS CONCEPTS

• **Social and professional topics** → **Computer science education**; **CS1**; • **Human-centered computing** → Synchronous editors.

KEYWORDS

pair programming, CS1, online learning, covid-19

ACM Reference Format:
Maxwell Bigman, Ethan Roy, Jorge Garcia, Miroslav Suzara, Kaili Wang, and Chris Piech. 2021. *PearProgram*: A More Fruitful Approach to Pair Programming. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21), March 13–20, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3408877.3432517

1 INTRODUCTION

Pair programming – where two programmers work together to write one program – is an industry and educational best practice with a long history of documented benefits for programmers of all levels [4] [43]. Although pair programming has been shown to have a number of benefits for introductory CS students at the K-12 and university levels [10] [39], the benefits tend to vary based on the contexts in which pair programming is implemented. CS education research has attempted to document some of these guidelines derived from classroom practice [44], however, there has been minimal evidence of implementation of rich insights from education research and the Learning Sciences to inform pair programming. Moreover, with the rapid shift to remote learning in the wake of the global pandemic in the Spring of 2020, guidelines for how to pair program need to be re-imagined for online learning environments.

In this paper we present *PearProgram*, an online collaborative tool to help introductory K-12 and university CS students learn how to pair program based on principles of Learning Sciences theory. Our main contributions are to: (1) present the importance of context in implementing pair programming; (2) introduce *PearProgram* and the theory behind it; (3) document the use of *PearProgram* in three online introductory CS courses; and (4) offer new insights on pair programming for practitioners, CS education researchers and learning scientists.

1.1 Related Work

Pair programming has become a best practice in CS education at the K-12 and university levels [30] [8]. Over two decades of work in CS education research, including two rigorous meta-analyses, have demonstrated that "if implemented properly" pair programming positively impacts students' programming assignment grades, exam scores, and persistence, especially in introductory CS classes [39] [9]. Other benefits of pair programming include greater confidence [15], deeper conceptual understanding [24], and continued interest in the topic [33] [42] [43].

1.2 A Bug in Pair Programming

Although numerous studies have highlighted the benefits of pair programming, less attention has been focused on the role that context and implementation have on the efficacy of the practice. We examined student performance in a recent offering of the CS1 course at our institution, a selective research university with a well-known CS department and a celebrated CS1 course. Students were given the option to pair program for homework assignments after the midterm exam, and of the 375 total students, 174 decided to work in pairs. Our analysis revealed that, on average, students who worked in pairs performed nearly identically to students who

SIGCSE '21, March 13–20, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8062-1/21/03…$15.00
https://doi.org/10.1145/3408877.3432517

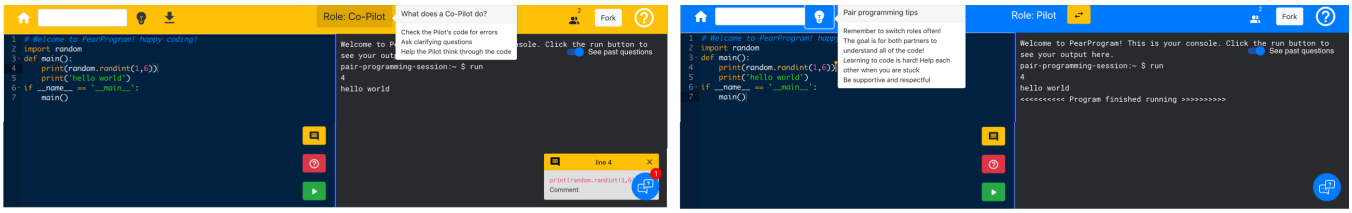

Figure 1: An example *PearProgram* session as seen from the copilot view (yellow) and pilot view (blue). The left-hand panel of the application is a text editor and the right is where the code output appears. The white text boxes at the top of each screen are some of the tips for pair programming embedded within the tool.

worked individually on homework assignments. Furthermore, students who worked individually scored higher on average than students who pair programmed ($d = 0.04$). Curiously, these findings suggest that pair programming does not necessarily help to improve performance in CS1 courses despite all of the studies demonstrating otherwise. This surprising result highlights that pair programming alone is not a sufficient pedagogical strategy, and raises questions around what supports might be necessary to realize the benefits of pair programming.

Indeed, we suspect that educators might overlook a key qualification that Umapathy and Ritzhaupt offered alongside their positive findings, namely that "students will need support and guidance in understanding the pair programming practice" [39]. Similarly, Williams guidelines for pair programming [44] leads with two recommendations: that "students need training on how to pair program," and that teaching staff should "actively manage" the pairs work together. Given that these conditions are not always possible, we seek to understand what happens when pairs work together under minimal guidance, and how learning tools might support best pair programming practices.

1.3 Changing Contexts: Remote Learners

While our research was underway, the onset of the global pandemic in March 2020 forced us to reconsider the form that our tool might take and the context in which our research could take place. Accordingly, we decided to shift our focus to remote learners. All of our corresponding design decisions were rooted in the paradigm of two students working together on the same program remotely. Indeed, there has been some work done on remote CS1 learners and "distributed pair programming" [14], and there is evidence that it can be just as effective as in-person pair programming [34]. While *PearProgram* can be used for in-person learning environments, we chose to explicitly design it for remote learners, recognizing both the challenges and opportunities that remote learning creates.

2 THEORETICAL FOUNDATIONS IN THE LEARNING SCIENCES

Pair programming as an educational practice occupies an interesting position: while it is commonly viewed as a CS education practice, it is nevertheless a specific type of collaborative learning. Indeed, the role of collaborative learning in K-12 education has been a popular research topic [18] that has been greatly informative to more recent Learning Sciences (LS) work on computer-based collaborative learning [33] [41]. The general finding from numerous studies on academic achievement has shown that students tend to learn better collaboratively than alone [25] [1] [38]. However, a more nuanced understanding of collaborative learning is needed. We adopt the view articulated by Dillenbourg et al. that "collaboration is in itself neither efficient nor inefficient. Collaboration works under some conditions, and it is the aim of research to determine the conditions under which collaborative learning is efficient" [11]. Thus, LS research stands to offer insights about how pair programming might produce the most "efficient" learning that leads to the best results for students, as well as what underlying mechanisms might contribute to more successful pairs. Our tool drew inspiration from the core tenets of the Learning Sciences: context, cognition and computation.

Context: There are important models from LS research of what contributes to successful collaborative problem solving: strong collaborations involve students actively discussing the task, sharing in the decision making process, building on one another's ideas, correcting each other's mistakes and resolving any conflicts that might arise [12] [16]. In effective collaborations, it is important that students view their success as mutually dependent, which leads to a dialogue that fosters new, co-created ideas and stands to establish a shared social reality [19] [31] [32]. Yet, benefits vary based on the context: the structure of the task (problem-oriented or not) and the composition of the pairs/groups are key variables that affect how successful a collaboration might be [17] [3]. In particular, our design choices stem from Barron's illumination of the complexities of collaborative work, theorized in terms of a "dual-space model" required for participants' collaborative problem solving, in which students need "to clarify how the content of the problem and the relational context are interdependent aspects of the collaborative situation" [3]. Prior research on pair programming has exclusively focused on the content of the problem and neglected the relational context, an area which the *PearProgram* tool brings to the foreground.

Cognition: Many studies in the Learning Sciences show the benefits of collaborative learning for both students' performance and also their cognitive development [2] [7]. Indeed, the critical role of discussion and reflection as students socially construct their learning has roots in education research on socio-constructivist theories of learning [40] . From a cognitive lens, working with a partner encourages students to explain their learning, respond to their partner's feedback, and work through challenging problems by asking questions and discussing what they do not understand – which all appear to improve achievement [6]. More recently, Pea's

theory of distributed cognition suggests that "knowledge is commonly socially constructed, through collaborative efforts toward shared objectives or by dialogues and challenges brought about by differences in persons' perspectives" [28]. Moreover, students must establish 'joint attention' [37] when working together, making it clear that both learners are coordinated in mutual engagement on the programming environment – a task which is made more complex in remote learning environments.

Computation: The CS Education literature has demonstrated the success of other collaborative learning strategies in computational learning. Three practices worth mentioning are: (1) contextualized computing, namely media computation [35]; (2) peer instruction [23, 29] ; and of course (3) pair programming (see above). CS Education research can largely be traced back to Seymour Papert and his theory of constructionism, which argued that the computer is a "universal construction machine" [27], or as he famously said, an "object-to-think-with" [26]. Papert's understanding of computation as a way of constructing knowledge "in the world" helps illuminate a bridge to the LS literature by highlighting the ways in which (pair) programming naturally affords learners the opportunity to create programs that spark a cyclical process of learning and computational creation.

3 INTRODUCING *PEARPROGRAM*

When we examined the available tools for remote pair programming, we found that none of them accomplished the specific set of tasks that we aimed for, namely: *a (standalone) user-friendly online collaborative IDE for introductory CS1 students to learn how to pair program*. Most notably, none of the tools that we explored explicitly taught students how to pair program nor did they support them in learning how to do so.

3.1 Design Aims

We set out to build a tool to accomplish three main goals:

(1) To help introductory CS students learn how to pair program effectively;
(2) To better understand how students currently pair program by capturing quantitative and qualitative data;
(3) To research what factors lead to different outcomes in different contexts.

3.2 Design Choices

It is worth noting that we occupy an interesting position in the CS education landscape: we are a team of learning scientists, education researchers, CS students and former CS teachers. Thus, we placed our diverse experiences as educators and learners in conversation with established theory from the Learning Sciences to inform a number of key design decisions for *PearProgram* outlined below.

Mutual Understanding is *the* Learning Goal: The implicit learning goal for students when pair programming is that they complete the programming task, individually learning the key concepts related to the problem they are working on. In *PearProgram*, we emphasize a different learning goal: that both students help each other to understand every line of code. Grounded in LS theory about the importance of mutual dependence, we feel that by making students responsible for one another's learning, they will be encouraged to help each other make sense of the program, and to not simply find the right answer or aim to finish the task as quickly as possible. One of the potential pitfalls of pair programming is that one student does a disproportionate amount of the work, by dominating the problem solving conversation and/or writing all of the code. By encouraging students to help one another understand each line of code, *PearProgram* aims to make students work together, communicate more, and teach one another.

Roles: One of the key features that distinguishes our tool from other online collaborative IDEs is that each programmer has a designated role with instructions on how to fulfill their responsibilities. We incorporate elements from the original guidelines for pair programming [44], such as restricting editing access so that only one programmer can write code at a given time. Furthermore, we want students to be deliberate about the code they are writing: if there is a suggestion, even simply fixing a syntax error, we want the partners to talk about that piece of code. We also extend the traditional conception of roles in pair programming, and deviate from the driver-navigator metaphor, choosing to use the terms *pilot* and *co-pilot* instead. Education research underscores the importance of roles and identity [20, 21], and we incorporated these ideas into *PearProgram* by giving students identities to assume and embedded tips on how to do so.

Embedded Teaching Tips: Often pair programming is introduced in a classroom setting with a set of slides and/or a short description of what to do (and not do) when pair programming. However, it can be hard for students to grasp and internalize the practice of pair programming in the abstract. In *PearProgram*, we decided to embed the teaching *into* the tool. Before starting a programming session, students watch a short video about pair programming and see the learning goals. Each role has specific instructions on how to fulfill their individual and shared responsibilities. Drawing on the idea of a *nudge* [36], students are prompted with helpful teaching tips approximately every ten minutes. These tips are grounded in pair programming best practices and the LS research, and include reminders to change roles, to ask questions, to listen to one another, to help one another and to be patient. In a future version, we plan to make these tips dynamically respond to the program and individualized for each partner.

Question & Comment Buttons: The Learning Sciences research highlights the importance of actively asking questions, engaging in conversation, taking turns and problem-solving. Thus, we created buttons in the *PearProgram* tool to encourage students to ask questions and annotate their code in real time. These buttons highlight particular lines of code, and provide visual cues to the partners: the comment button simply highlights the code, while the question button creates a red border to encourage students to stop and talk about the question. The use of these features feels particularly important for creating a "joint attention space" – directing students' attention to the same part of the code – which is a key factor for establishing distributed cognition, especially in a remote learning environment.

3.3 *PearProgram* Workflow

The intended use for *PearProgram* is as a CS student's first exposure to pair programming. The tool (see Figure 1) has lessons integrated

into the tool, including introductory videos, embedded teaching tips during the programming session, and features designed for students to actively engage with the practice of pair programming. As originally envisioned, a teacher would create sessions for pairs of students to work together, send them each the relevant links to the collaborative *PearProgram* session, students would log on, and (ideally) be engaging in a video or audio conversation (e.g. on Zoom) while working together. (The tool currently has a text-based chat feature, and a future version will hopefully have full video integration). Students are then engaged in a synchronous, remote pair programming environment, both seeing the same IDE as well as where their partner is working, with the ability to ask questions, leave comments and switch roles.

4 HOW STUDENTS USED *PEARPROGRAM*

We used *PearProgram* in three different online CS1 courses. The first, *Code in Place*, was a large-scale, open access teaching experiment taught in the midst of the COVID-19 pandemic to over 10,000 students around the world. In *Code in Place*, we used *PearProgram* both in small group discussion sections, as well as in one informal office hours in which we paired students off to work together. The second use of *PearProgram* was in our institution's CS1 course, in which it was used across a number of discussion sections. The third use of *PearProgram* was in *CS Bridge*, an international, multi-institution three-week course to teach high school students the fundamentals of computer programming. In *CS Bridge* all of the roughly 175 students had the opportunity to use *PearProgram* for a small, unsupervised programming project with access to optional help from TAs.

4.1 Interactions with *PearProgram*

Table 1: Summary statistics of different *PearProgram* features for the 51 sessions used in the online high school class

Feature	Mean Usage	St. Dev.
Chat	48.0	42.3
Code Output	8.2	7.0
Comment	1.5	6.6
Confusion Button	0.1	0.3
Resolve Confusion	0.03	0.2
Pilot Hand-off	2.7	2.7
Toggle Request	2.9	3.1

During *CS Bridge*, the online introductory CS course geared towards high school students, learners used *PearProgram* to write an interactive console program. Students had recently covered variables in the course and this activity allowed them to practice using and updating variables in a text based setting. After students completed the activity, we examined data generated by *PearProgram* to better understand how students engaged with the various features embedded within the tool. We explored the use of the chat, run code, comment, confusion, resolution, toggle request and role change features in the 31 pair programming sessions, as well as the temporal dynamics (Figures 3a and 3b). Summary statistics of these features (Table 1) revealed that students used the chat, run

Figure 2: Visualization of each session and feature use within each session. The x-axis represents the length of each session and the y-axis represents each individual session. The color of each bar corresponds to which partner was in the pilot role at a given time. Overlaid on each session are shapes corresponding to the various features

and toggle features frequently, but largely ignored the comment and confusion features. Meanwhile, Figure 2 maps the length of each session, which partner is in the pilot role, and when within each session either partner sent a chat, ran the code, or used the question or comment features

5 TEACHER ENTHUSIASM

We spoke to a number of teaching assistants (TAs) who used *PearProgram* to gather insights about what worked well in their classes.

TAs benefited from the ability to simultaneously observe all of their students coding at once. Unlike during in-person discussion sections where TAs can be physically present with student pairs listening to the dialogue and moving around the room to help students, the online version of discussion section limits transparency. In particular, *Zoom* breakout rooms mean that the TA can at most help one pair of students at a given time without access to any other students. One of the biggest takeaways from TAs is how much they appreciated the ability to watch all of their students' coding sessions simultaneously on the *PearProgram* tool. As one TA summarized, "the best part of using *PearProgram* was the fact that I was looking at all of their workspaces," which allowed TAs to be able to move seamlessly from one group to the next and offer targeted help. TAs also liked being able to leave comments and suggestions for students without ever having to enter the breakout room, as explained in more depth below.

TAs left comments for students while they were working. All of the TAs we spoke with mentioned how they liked the tool's comment feature as a way to leave feedback while students were working without interrupting their workflow. All of the TAs used the comment feature extensively to help their students through challenges, although how they used it varied slightly. One TA went into her students' sessions to see if they were on the right track,

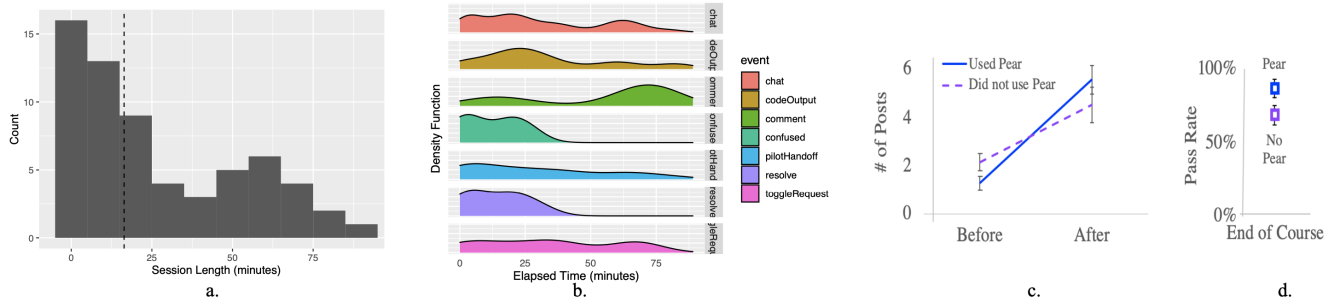

Figure 3: (a) Histogram of session length. Dotted line represents the median. (b) Density plot of feature usage over 90 minutes. (c) μ change in discussion posts before and after intervention (d) Completion rates. Standard error bars.

leaving comments and hints if she found mistakes, as well as celebrating good decisions the pairs had made. Another TA mentioned how she liked to "gently nudge" her students towards the correct approach, leaving generative questions to spark discussions amongst the pairs. A third TA stated that her aim with the comments was to "show students direction" rather than telling them directly what to do, sparking conversations about what did and did not work in the code and why.

TAs found the tool very useful for group coding sessions. Part of our institution's model for CS courses is to have smaller weekly discussion sections with roughly 10-15 students. Two of the TAs we spoke to used the *PearProgram* tool to create group collaborative coding sessions in their discussion sections which created an opportunity for a dynamic live coding session. Both TAs mentioned how useful it was for their students to all be able to seamlessly take on the *pilot* role, as well as the ability to anonymously leave both questions and comments on the code – which seemed to create more active engagement from their students. One TA specifically liked the visual cue of questions, and she spoke to the importance of anonymity as this allowed students to ask questions that they might not feel comfortable asking in front of the whole class.

Students embraced new identities. Analysis of qualitative data and case studies surrounding *PearProgram* also illustrated that the tool may play a role in influencing learner's attitudes towards their own identities as programmers. One powerful anecdote shared by a TA was of a student who expressed uncertainty about her identity as a programmer and considered dropping out of the CS1 course prior to pair programming. However, she became empowered by roles of "pilot" and "co-pilot" embedded within the tool. Though this change in attitude and increase in confidence may stem from the act of pair programming, the tool's explicit use of two active roles as opposed to the traditional, more passive, drive-/navigator roles may play a role in shaping learner's identities both within their pairs and as programmers.

6 IMPACT ANALYSIS

Through these early experiences we have gathered interesting and promising findings for future development and research.

6.1 Students and Teachers liked *PearProgram*

We surveyed student users of *PearProgram* (n= 57) and found that 74% of them felt that the tool taught them how to pair program, 93% of the students enjoyed the experience, and 70% of the students would want to use it again. Furthermore, 72% felt more confident programming after using the tool and 53% of the students felt that they were better teachers after using the tool. We also surveyed the teachers using the tool (n=26) and found that 100% of them wanted to use the tool again. It is important to note that neither of these surveys were anonymous.

6.2 Students who used *PearProgram* did better

After the online high school CS1 course finished, we examined the relationship between use of *PearProgram* and two learning outcomes: course engagement and course pass rates.

Course Engagement Increased: Of the 155 students that took the course, 82 students used *PearProgram* to work collaboratively on an assignment and 73 worked individually for the duration of the course. We first looked at levels of course engagement, as measured by number posts in the course online discussion forum before and after the *PearProgram* activity. This analysis showed that students who pair programmed had a significant increase in the number of discussion forum posts after using *PearProgram* compared to students who worked alone (Figure 3c).

Higher Pass Rates: We were also able to examine the course pass rates for students who used *PearProgram* and those who did not. This analysis found that across the whole class, 130 of 155 students passed the course (pass rate = 84%). When broken down by those who pair programmed and those who did not, 75 of the 82 students who pair programmed passed the class (pass rate = 91%), whereas 55 of the 73 students who did not pair program passed the class (pass rate = 75%). As seen in Figure 3d, the effect size for this difference was about 16 percentage points ($p = 0.007$).

Limitations: It should be noted that this was not a randomized control experiment and these results should in no means be interpreted as causal. One important potential confound is that students who were already disengaging from the course before the introduction of *PearProgram* did not feel compelled to interact with the tool and simply continued to disengage after the *PearProgram* activity,

resulting in fewer forum posts and lower pass rates. Despite the current inability to make any causal claims, we did observe a positive correlation between use of *PearProgram* and learning outcomes, as measured by discussion engagement and passing rates.

7 DISCUSSION: LESSONS LEARNED

7.1 Implications for CS Education

The promising initial results from use of *PearProgram* create opportunities for CS instructors to implement pair programming in their (introductory) classes without having to explicitly teach students how to pair program. While there are limitations to the initial study, students who engaged with the *PearProgram* tool appeared to realize many of the benefits promised by pair programming: namely, increased performance [39], greater confidence [15], as well as higher levels of enjoyment [5]. *PearProgram* thus alleviates the burden from instructors of needing to teach the practice of pair programming, and it also offers interesting data points about students that can be used to create more personalized learning experiences. Specifically, *PearProgram* generates information such as who wrote the bulk of the code, who prefers to be driver or navigator, which pairs had the most success, and how long different pairs took to complete a task (see Figure 2). These insights from the *PearProgram* tool allow for new understanding for instructors, TAs and researchers. In particular, as CS instruction is increasingly forced to move to remote environments, it creates new possibilities for students to continue to engage with pair programming even when they are not co-located.

Another exciting application area for instructors is in supervised lab work, especially in remote settings. Given the initial success of *PearProgram* in TA-led discussion sessions, it seems that the tool offers new possibilities for in-person and remote TAs to work with many pairs of students at a given time, similar to what Guo has been able to accomplish with *Codeopticon* where one teacher can monitor as many as hundreds of learners at a time [13]. Moreover, it creates opportunities for group coding sessions in remote discussion sections and lab environments.

7.2 Implications for CS Education Research

As discussed above, meta-analyses of pair programming suggest that in order to realize the benefits of the practice it must be "implemented properly," which includes the fact that "students will need support and guidance in understanding the pair programming practice" [39]. Given its explicit design as both a learning and research tool, *PearProgram* offers a new way of understanding what conditions might be most successful for students when pair programming, and presents the possibility for a combination of qualitative and quantitative analysis to uncover the "black box" of what makes pair programming work. Future research can explore student attitudes, prior skill levels, video analysis of pair interactions, and the coding session itself to gain a deeper understanding of what conditions lead to the most successful pair programming experiences. Of particular interest to the CS education research community might be questions of how to best pair students, what makes "distributed" (what we refer to as 'remote') pair programming successful, why pair programming works for some students but not others, who pair programming benefits the most, as well as what long term benefits pair programming might offer to students as they move through their CS courses. Additionally, *PearProgram* and the broader shift to remote instruction suggest that after a dozen years it might be time to revisit the 'Eleven Guidelines' for pair programming [44] and update them with more recent findings from both LS theory and practice, as well as for new contexts (such as remote learners). Similarly, the driver-navigator roles might be another area worth investigating and updating, as our initial findings and education research suggest that roles and labels influence student identity and engagement with the materials. Finally, the potential benefits that the Learning Sciences offer to CS education research suggest that the two fields might create synergistic effects that stand to benefit CS learners, teachers and researchers – and might be a fertile area of exploration for future research [22].

7.3 Implications for Learning Sciences Research on CS

As learning scientists focused on CS education, we are drawn to three particular sets of factors for further inquiry: contextual, cognitive and computational.

Context: Our observations of pair programmers using *PearProgram* indicated unique communication patterns cross-culturally and within gender. How do different students' socio-cultural and economic realities impact the nature of learning in a pair programming context? Might there be unique benefits to underrepresented students' learning accessed by the roles made available? How do questions of stereotype threat, positioning and power play out in CS learning environments?

Cognition: Interviews with *PearProgram* users indicate a shift in attitudes towards listening and enhanced agency in their learning. Is material learned through an arduous solitary process comparable to material learned in an enjoyable cooperative process? Do the secondary skills acquired via collaboration (listening, concise communication, respectful ideation) transfer to other learning contexts? What affordances and limitations does pair programming offer students in establishing a joint attention space [32], particularly in remote learning environments?

Computation Our observations suggest that students engage the available communication and "help" features in unique ways. To what extent can the tool serve as a learning coach or instructor of collaborative practices? Is the tool enabling a more comprehensive understanding of coding languages? What might the role of teaching tools be in facilitating collaborative learning practices in CS education?

8 CONCLUSION

Pair programming is an important CS education practice that stands to benefit from a deeper understanding of the contextual factors of when it works best, and how educators might support its implementation for students to realize its numerous benefits. Utilizing key insights from Learning Sciences, *PearProgram* appears to be a promising tool that can both support (remote) introductory CS students, and also reveal promising and important research findings about pair programming. Further use of *PearProgram* in a wider variety of settings will help inform the development of both the tool and future research.

REFERENCES

[1] Philip C Abrami and Bette Chambers. 1996. Research on cooperative learning and achievement: Comments on Slavin. *Contemporary Educational Psychology* 21, 1 (1996), 70–79.

[2] Brigid Barron. 2000. Achieving coordination in collaborative problem-solving groups. *The journal of the learning sciences* 9, 4 (2000), 403–436.

[3] Brigid Barron. 2003. When smart groups fail. *The journal of the learning sciences* 12, 3 (2003), 307–359.

[4] Kent Beck. 2000. *Extreme programming explained: embrace change*. addison-wesley professional.

[5] Cathy Bishop-Clark, Jill Courte, and Elizabeth V Howard. 2006. Programming in pairs with Alice to improve confidence, enjoyment, and achievement. *Journal of educational computing research* 34, 2 (2006), 213–228.

[6] Ann L Brown, Annemarie S Palincsar, and Bonnie B Armbruster. 1984. Instructing comprehension-fostering activities in interactive learning situations. *Learning and comprehension of text* (1984), 255–286.

[7] CK Chan. 2013. Towards a Knowledge Creation Perspective. *The international handbook of collaborative learning* (2013), 437.

[8] K CSTA. 12. Computer science standards. *Computer Science Teachers Association* (12).

[9] Carolina Alves de Lima Salge and Nicholas Berente. 2016. Pair Programming vs. Solo Programming: What Do We Know After 15 Years of Research?. In *2016 49th Hawaii International Conference on System Sciences (HICSS)*. IEEE, 5398–5406.

[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] P Dillenbourg, M Baker, A Blaye, C O'Malley, and E Spada. 1996. Learning in humans and machines: Towards an interdisciplinary learning science. *The evolution of research on collaborative learning* (1996), 189–211.

[12] Merrilyn Goos, Peter Galbraith, and Peter Renshaw. 2002. Socially mediated metacognition: Creating collaborative zones of proximal development in small group problem solving. *Educational studies in Mathematics* 49, 2 (2002), 193–223.

[13] Philip J. Guo. 2015. Codeopticon: Real-Time, One-To-Many Human Tutoring for Computer Programming. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software and Technology (UIST '15)*. ACM, New York, NY, USA, 599–608. https://doi.org/10.1145/2807442.2807469

[14] Brian Hanks. 2005. Student Performance in CS1 with Distributed Pair Programming. *SIGCSE Bull.* 37, 3 (June 2005), 316–320. https://doi.org/10.1145/1151954.1067532

[15] Brian Hanks, Charlie McDowell, David Draper, and Milovan Krnjajic. 2004. Program quality with pair programming in CS1. In *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*. 176–180.

[16] Cindy E Hmelo-Silver and Howard S Barrows. 2008. Facilitating collaborative knowledge building. *Cognition and instruction* 26, 1 (2008), 48–94.

[17] Learning Cindy E Hmelo-Silver and Christina DeSimone. 2013. Problem-based learning: An instructional model of collaborative learning. In *The international handbook of collaborative learning*. Routledge, 382–398.

[18] David W Johnson and Roger T Johnson. 1999. Making cooperative learning work. *Theory into practice* 38, 2 (1999), 67–73.

[19] David W Johnson and Roger T Johnson. 2004. Cooperation and the use of technology. (2004).

[20] Jennifer M Langer-Osuna. 2016. The social construction of authority among peers and its implications for collaborative mathematics problem solving. *Mathematical Thinking and Learning* 18, 2 (2016), 107–124.

[21] Jennifer M Langer-Osuna. 2017. Authority, identity, and collaborative mathematics. *Journal for Research in Mathematics Education* 48, 3 (2017), 237–247.

[22] Lauren E. Margulieux, B. Dorn, and Kristin A. Searle. 2019. Learning Sciences for Computing Education.

[23] Eric Mazur. 2013. Peer instruction. (2013).

[24] Charlie McDowell, Linda Werner, Heather E Bullock, and Julian Fernald. 2006. Pair programming improves student retention, confidence, and program quality. *Commun. ACM* 49, 8 (2006), 90–95.

[25] Angela M O'Donnell and Donald F Dansereau. 1992. Scripted cooperation in student dyads: A method for analyzing and enhancing academic learning and performance. *Interaction in cooperative groups: The theoretical anatomy of group learning* (1992), 120–141.

[26] Seymour Papert. 1980. Mindstorms: Computers, children, and powerful ideas. *NY: Basic Books* (1980), 255.

[27] Seymour Papert and Idit Harel. 1991. Situating constructionism. *Constructionism* 36, 2 (1991), 1–11.

[28] Roy D Pea. 1993. Practices of distributed intelligence and designs for education. *Distributed cognitions: Psychological and educational considerations* 11 (1993), 47–87.

[29] Leo Porter, Dennis Bouvier, Quintin Cutts, Scott Grissom, Cynthia Lee, Robert McCartney, Daniel Zingaro, and Beth Simon. 2016. A multi-institutional study of peer instruction in introductory computing. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. 358–363.

[30] Leo Porter, Mark Guzdial, Charlie McDowell, and Beth Simon. 2013. Success in Introductory Programming: What Works? *Commun. ACM* 56, 8 (Aug. 2013), 34–36. https://doi.org/10.1145/2492007.2492020

[31] Barbara Rogoff. 1990. *Apprenticeship in thinking: Cognitive development in social context*. Oxford university press.

[32] Jeremy Roschelle and Stephanie D Teasley. 1995. The construction of shared knowledge in collaborative problem solving. In *Computer supported collaborative learning*. Springer, 69–97.

[33] R Keith Sawyer and Kenneth J Goldman. 2010. Collaborative learning of computer science concepts. *Educational dialogues: Understanding and promoting productive interaction* (2010), 323–345.

[34] Julia Schenk, Lutz Prechelt, and Stephan Salinger. 2014. Distributed-pair programming can work well and is not just distributed pair-programming. In *Companion Proceedings of the 36th International Conference on Software Engineering*. 74–83.

[35] Beth Simon, Päivi Kinnunen, Leo Porter, and Dov Zazkis. 2010. Experience report: CS1 for majors with media computation. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*. 214–218.

[36] Richard H Thaler and Cass R Sunstein. 2009. *Nudge: Improving decisions about health, wealth, and happiness*. Penguin.

[37] Michael Tomasello et al. 1995. Joint attention as social cognition. *Joint attention: Its origins and role in development* 103130 (1995).

[38] SS Totten, TA Digby, and P Russ. 1991. Cooperative Learning: A Guide to Research. new York: garland.

[39] Karthikeyan Umapathy and Albert D Ritzhaupt. 2017. A meta-analysis of pair-programming in computer programming courses: Implications for educational practice. *ACM Transactions on Computing Education (TOCE)* 17, 4 (2017), 1–13.

[40] Lev Vygotsky. 1978. Interaction between learning and development. *Readings on the development of children* 23, 3 (1978), 34–41.

[41] Rupert Wegerif. 2010. *Mind expanding: teaching for thinking and creativity in primary education: Teaching for Thinking and Creativity in Primary Education*. McGraw-Hill Education (UK).

[42] 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–es.

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

[44] Laurie Williams, D Scott McCrickard, Lucas Layman, and Khaled Hussein. 2008. Eleven guidelines for implementing pair programming in the classroom. In *Agile 2008 Conference*. IEEE, 445–452.