Think First: Fostering Substantive Contributions in Collaborative Problem-Solving Dialogues

Mehmet Celepkolu, Joseph B. Wiggins, Kristy Elizabeth Boyer, Kyla McMullen
University of Florida, Gainesville, FL, USA
mckolu@ufl.edu, jbwiggi3@ufl.edu, keboyer@ufl.edu, drkyla@ufl.edu

Abstract: Working collaboratively holds many benefits for learners. However, varying incoming knowledge and attitudes toward collaboration present challenges and can lead to frustration for students. An important open question is how to support effective collaboration and foster equity for students with different levels of incoming preparation. In this study, we compared two collaborative instructional approaches for computer science problem solving, in which students participated in one of two conditions: The *Baseline* condition featured collaborative problem solving in which students worked in dyads from the beginning of the collaboration; in the other condition, called *Think-First*, students first worked on the problem individually for a short time and then began collaborating to produce a common solution. The results from 190 students from an introductory programming class working in 95 pair-programming teams demonstrate that this simple modification to pair programming had a significant positive effect on test scores and on substantive contributions in collaborative dialogue.

1. Introduction

The CSCL community has long worked to understand important facets of collaboration, such as the importance of sharing and constructing knowledge together (Stahl, 2002; Ludvigsen et al., 2011). A central idea that has emerged from this research is that explicit and/or implicit structure for learning activities can significantly improve the quality of collaboration (Beers et al., 2007; Kirschner et. al, 2008). One example of a collaboration structure that has been actively used in the domain of computer science is *pair programming*, in which two learners work side by side at the same computer and collaborate on the same code. While one of the learners, the *driver*, is typing at the computer, the other partner, the *navigator*, assists in planning, strategizing, and pointing out errors. Through pair programming, students produce higher quality code, derive greater enjoyment from the programming experience, and learn more from each other compared to individual programming (Cockburn & Williams, 2000; Dybå et al., 2007; McDowell et al., 2002; Nagappan et al., 2003; Preston, 2006; Williams et al., 2000). Previous research has also provided insight into how students should be paired together, using approaches such as personality tests and previous test scores (Chao & Atli, 2006; Werner et al., 2004). However, there has been very little investigation of how to support collaboration after students are paired.

When individual learners within a dyad have substantially different levels of knowledge or motivation, support for collaboration is particularly important. For example, when students with different knowledge levels work on the same problem, information usually flows from the *high-performing* student to the *low-performing* student rather than through a balanced dialogue (McCarthey & McMahon, 1992). Similarly, some students, when paired with a less knowledgeable peer, can feel that collaboration is not worthwhile (Bevan, Werner & McDowell, 2002). These challenges highlight a pressing open research question: ***How can we effectively support collaborative problem-solving dialogue between two students with contrasting levels of incoming preparation?***

To investigate this research question, we conducted a study of collaborative learning for computer science, in which there were two conditions: one in which students collaborated in standard pair programming (the *Baseline* condition), and one in which students worked on the problem individually for fifteen minutes and then began collaborating with their partner to construct a common solution (the *Think-First* condition). We hypothesized that students in the *Think-First* condition would achieve higher learning outcomes than students in the *Baseline* condition while reporting equal or higher levels of enjoyment. The results support these hypotheses. Without the *Think-First* structure, stronger students often started out more prepared and took increasing control, while the less prepared students became increasingly marginalized as time went on. The results show that reserving some time for students to think individually before they began to collaborate improved learning outcomes without adding extra total time, and without negatively affecting their enjoyment of the collaborative process.

2. Prior Work

A widely used instructional design approach is for students to be given some initial time to engage with the learning activity individually and then be expected to work in a small-group or share their knowledge with the whole-class. This approach has been shown to increase engagement in computer science courses (Fitzgerald, 2013; Kothiyal et al., 2013) and improve student achievement (Kaddoura, 2013; Siburian, 2013; Sugiarto & Sumarsono, 2014; Azlina & Nik 2010). This instructional method can potentially help computer science courses as it gives students time, which allows them to judge what they know, prepares them for collaboration, and eventually improves their higher-order thinking skills (Yerigan, 2008). Examining the dialogues between dyads can shed light on the underlying phenomena that lead to success in collaborative learning, such as negotiation of meaning and construction of shared conceptions (Stahl, Koschmann & Suthers 2006). Stahl (2006) states "meaning is created across the utterances of different people." Studies of spoken or textual dialogues help us to understand the changing needs of individuals in dyads or groups (Davidsen & Ryberg, 2015).

Socio-Constructivist theory suggests that learning is a social, interactive and collaborative process through which students develop higher-order thinking skills, such as reasoning and problem solving (Kozulin et al., 2003). Vygotsky (1978) emphasizes the significance of pairs remaining within each other's "Zone of Proximal Development" (ZPD) because a large difference can create too much or too little challenge, leading to confusion or demotivation. Matching students with partners who operate within their own ZPD is very important; however, practical challenges in classrooms usually make it difficult to match students perfectly in this way. Structured support for collaborative problem solving can be helpful even in the presence of contrasting incoming knowledge levels. In this study, we examine the simple strategy of setting aside time for individual thinking before collaboration begins.

3. Methods

We conducted a quasi-experimental study to investigate differences between the *Baseline* and *Think-First* conditions and examined case studies to make qualitative observations. Pre- and post-surveys and content knowledge posttests were analyzed using descriptive and inferential statistics. We also examined five Baseline and five Think-First dyads' collaborative learning activities during pair programming, which we video recorded and transcribed manually.

3.1. Participants

The study was conducted with students who were actively enrolled in the first computer science class for computer science majors at a university in the southeastern United States. The class was taught in the Java programming language and had a total enrollment of 471. Class met three times per week in a large lecture hall. In addition to lectures, each student attended one two-hour lab per week in which they collaboratively solved lab exercises. Undergraduate and graduate teaching assistants led these labs. All students completed the same lab exercises (including students who were not part of the study). We only collected data on the 190 students who consented to participate in research, which was voluntary. Of the 190 students, there were 85 freshmen (45%), 56 sophomores (29%), 41 juniors (21%), 5 seniors (3%), and 3 with other designations (2%). 125 students (66%) reported having prior programming experience and 91 (48%) reported prior experience in the Java programming language. There were 46 (24%) women and 144 (76%) men. Of all participants, 54% identified as White/Caucasian, 17% as Hispanic/Latino, 14% as Asian/Pacific Islander, 4% as Black/African-American, and 11% as Other (including multiracial and some races not listed in the above categories).

One week prior to the study, students took a midterm exam as part of regular course activities. The average score on the exam was 75.6 (*median* = 77; *stdev* = 15.2). We grouped consenting students based on whether they were above the median (High) or below the median (Low). Students were assigned into "High-High", "High-Low" and "Low-Low" dyads randomly. We used the midterm scores as the pre-test to gauge levels of understanding before the lab to compare to the post-test after the lab. There are no significant pre-test differences between the *Baseline* and *Think-First* conditions (High performing students in High-High dyads $p = 0.2370$; high performing students in High-Low dyads $p = 0.1302$, Low performing students in High-Low dyads $p = 0.7068$ and Low performing students in Low-Low dyads $p = 0.4331$). After pairing was complete, each lab section (consisting of between 5 to 10 dyads) was randomly assigned to either the *Baseline* or the *Think-First* condition.

Table 1. Number of dyads in each category considered in this study.

Dyad Type	*Baseline* Condition	*Think-First* Condition	TOTAL
High - High	15	17	32 pairs
High - Low	20	11	31 pairs
Low - Low	17	15	32 pairs
TOTAL	52	43	**95 pairs**

3.2 Procedure

Students participated in the study as part of their regular two-hour lab section. At the beginning of the lab, students were placed into their pre-arranged dyads. All dyads completed a learning task involving one-dimensional arrays. In the *Baseline* condition, students began pair programming immediately. In the *Think-First* condition, students worked individually for fifteen minutes and then began pair programming to construct a collaborative solution. After they completed the assignment, or when the allocated time was over (two lab hours), students individually completed a ten-question pair programming attitude survey. Then, they completed a posttest consisting of ten multiple-choice questions on one-dimensional arrays. We randomly selected five dyads from each condition to be video and audio-recorded throughout their collaboration.

4. Results and Discussion

We compared learning outcomes in the *Baseline* and *Think-First* conditions. The results show that students performed significantly better on the posttest in the *Think-First* than in the *Baseline* condition. This result held for both low-performing and high-performing students (as measured on the preceding midterm exam). As shown in Figure 2, high-performing students in the *Baseline* condition had a mean posttest score of 4.22 (n = 50; *stdev* = 2.24) and high-performing students in the *Think-First* condition had a mean posttest score of 5.20 (n = 45; *stdev* = 1.94). This difference is significant (p = 0.0257, ANOVA). For only the high-performing students, we examined the effect of the *Think-First* condition on different dyad types. The effect of *Think-First* was not significant for high-performing students paired with other high-performing students, but high-performing students paired with low-performing students achieved significantly higher posttest scores (p = 0.0429, Wilcoxon rank-sum test) in the *Think-First* condition compared to *Baseline*. Table 2 shows the outcomes for high-performing students.

Low-performing students also clearly benefitted from the *Think-First* condition. As shown in Figure 2, low-performing students in the *Baseline* condition had a mean posttest score of 3.74 (n = 54; *stdev* = 2.25) and those in the *Think-First* condition had a mean posttest score of 5.12 (n = 41; *stdev* = 2.24). This difference is also significant (p = 0.0038, ANOVA). For low-performing students paired with another low-performing student, the effect of *Think-First* was significantly positive (p = 0.012 Wilcoxon rank-sum test). The benefit for low-performing students paired with high-performing students was not statistically significant (p = 0.16). Table 3 shows the outcomes for low-performing students.

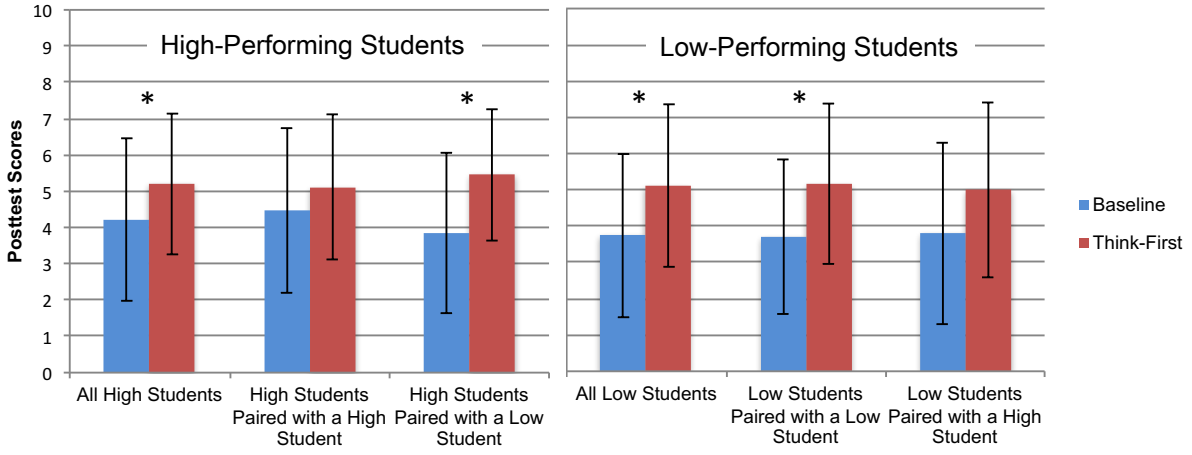

Figure 2. *Think-First* effect on students' individual posttest score by collaborative condition (* indicates significant difference between *Baseline* and *Think-First* conditions)

Table 2. Summary of pre-test scores and outcomes for high-performing students

	High-performing students paired with another High-performing student mean (stdev)		***High-performing students paired with a Low-performing student*** mean (stdev)	
	Baseline **(n = 30)**	***Think-First*** **(n = 34)**	***Baseline*** **(n = 20)**	***Think-First*** **(n = 11)**
Preceding Midterm (out of 100)	88.7 (6.34)	86.97 (5.07)	86.7 (6.07)	90.45 (6.47)
Post Test (out of 10)	4.47 (2.27)	5.12 (2.00)	3.85 (2.21)	5.45 (1.81)
Enjoyment (out of 50)	37.20 (10.16)	35.06 (9.25)	37.50 (8.79)	31.72 (11.13)

Table 3. Summary of pre-test scores and outcomes for low-performing students

	Low-performing students paired with another Low-performing student mean (stdev)		***Low-performing students paired with a High-performing student*** mean (stdev)	
	Baseline **(n = 34)**	***Think-First*** **(n = 30)**	***Baseline*** **(n = 20)**	***Think-First*** **(n = 11)**
Preceding Midterm (out of 100)	60.64 (10.91)	62.93 (12.10)	67.05 (10.97)	65.18 (14.00)
Post Test (out of 10)	3.71 (2.13)	5.17 (2.21)	3.80 (2.50)	5.00 (2.41)
Enjoyment (out of 50)	38.06 (8.31)	39.83 (9.90)	35.90 (11.54)	39.55 (10.23)

Our hypothesis was that students in the *Think-First* condition would achieve higher learning outcomes compared to students in the *Baseline* condition. Indeed, this was the case. *Low-performing* students benefitted more overall: there was a medium effect size for the low-performing students (Cohen's $d = 0.59$) and a small effect size for the high-performing students (Cohen's $d = 0.46$).

To explore the ways in which the *Think-First* condition influenced collaborative problem-solving dialogue compared to the *Baseline* condition, we now consider several excerpts. Figure 3 left shows a conversation between a low-performing student (whom we call Lindsay) and a high-performing student (whom we call Hector). In this interaction, Lindsay took on the role of driver first, and she began writing code. As soon as Lindsay began coding, the pair asked questions of each other. But soon, Lindsay made mistakes and Hector corrected her several times in less than one minute. Within two minutes of working on the programming code, Lindsay asked Hector if he wanted to be the driver. Hector accepted the offer and took control. This example from the *Baseline* condition illustrates a case of a less-prepared student losing confidence and giving up control quickly. In typical programming, we see drivers and navigators switch roles every fifteen to thirty minutes, but in Figure 3 left, Lindsay gave up control after less than two minutes of active coding.

We contrast Hector and Lindsay's interaction with that of a *High-Low* dyad from the *Think-First* condition. Figure 3 right shows dialogue between Luke (a low-performing performing student) and Hilda (a high-performing student). Hilda began as the driver and Luke contributed to the process of code writing. As soon as the collaboration started, Luke made a suggestion about how to start solving the problem, and Hilda agreed. At the 04:41 mark, Luke

referred to a previous lab assignment, which he had looked up individually during the *Think-First* time. Although he was the low-performing student in the dyad, he was able to make this substantive contribution because he used the *Think-First* time to refresh his memory and prepare for collaboration. At the 05:13 mark, Hilda made a suggestion and Luke responded by accepting it. After this step, Hilda offered for Luke to do the next step and Luke agreed. In contrast to Lindsay in the *Baseline* condition, who was eager to give up control quickly, Luke accepted control willingly.

Baseline session	***Think-First session***
Lindsay (06:10) Ok, we could take an input first. **Hector (06:13)** No, I think you need a scanner, right? **Lindsay (06:15)** Yes, new scanner? **Hector (06:17)** Not scanner. I use Command key + B. It is like keyboard **Lindsay (06:23)** Alright, let's call it console, do you care? **Hector (06:25)** No. **Lindsay (06:25)** I am calling it console. Console equals to new? I am definitely doing it wrong. **Hector (06:30)** No, it is new, it is true. New scanner. Create a new object so you are making a new scanner object and then like whatever parameter just system.in … **Lindsay (07:54)** Do you wanna do this? (Offering Hector to become driver.)	**Luke (04:33)** So I guess the first part is the random number. **Hilda (04:40)** Yeah **Luke (04:41)** we need to, although... So I looked up from our last lab, we could make random number by saying randNumber. They say they call it randnumber **Hilda (04:51)** Yeah **Luke (04:52)** So if you do randnumber = math.random **Luke (05:00)** Capital A. You have to do Math. Capital M. Uh… Times one thousand **Hilda (05:08)** So, that is on the outside **Luke (05:11)** and we have not imported it yet so it is gonna give an error. **Hilda (05:13)** Alright. Let's just do that. However it is an Int, I guess **Luke (05:18)** Yeah it will be an int. We haven't imported it yet so we gotta go top to do import **Hilda (05:24)** Ok. Do you want to do this part? **Luke (05:26)** Yes, sure

Figure 3: Illustrative excerpts from *High-Low* dyad in a *Baseline* condition on the left and a *High-Low* dyad from the *Think-First* condition on the right

The previous two excerpts illustrate collaboration between *High-Low* dyads. Next we examine dialogues between *Low-Low* dyads. Figure 4 left shows a conversation between Lloyd (*low*-performing) and Larry (*low*-performing) in the *Baseline* condition. Larry started as the driver and Lloyd first suggested creating a new Scanner (to receive user input) but Larry was unaware of how to do this, so he asked what they needed. Larry continued typing, but he was not sure how to import a scanner, so he asked Lloyd to clarify it. At 07:35, Larry said they needed to create a scanner and Lloyd told him that he usually names it "input". However, Larry struggled to create the scanner and Lloyd ultimately dictated the syntax to Larry. Lloyd was visibly frustrated and becoming increasingly disengaged.

We contrast Lloyd and Larry's experience with a *Low-Low* dyad from the *Think-First* condition. Figure 4 right shows a conversation between Laura (*low-performing*) and Lewis (*low-performing*) in the *Think-First* condition. At first, Laura and Lewis agreed on creating a Scanner, but, like Lloyd and Larry, they did not recall how to do it. However, they spent two minutes checking their own notes made during the Think-First time, and collaborated productively to complete the needed line of code. Neither student appeared to become frustrated during this time, and they both made substantive contributions as they worked through the challenges they faced.

Baseline session	*Think-First session*
Lloyd (06:07) We need to import scanner first.	**Laura (03:11)** Alright. First thing first.. Oh yeah, we need a scanner
Larry (06:10) We need what?	**Lewis (03:12)** Hmm hmm
Lloyd (06:12) Import scanner	**Laura (03:31)** Oh, ok. Hmmm. How does that one go there?
Larry (06:13) Oh, you are right. Java utils and... Is it space scanner or dot scanner?	**Lewis (03:32)** Scanner console equals System dot in?
Lloyd (06:23) Dot scanner	**Laura (03:43)** I think there was something else too
Larry (06:24) oh ok	**Lewis (03:45)** Yeah there is something before
Lloyd (06:26) Also, uppercase	**Laura (03:47)** Is it a "new" or something?
Lloyd (07:05) … so the first one should probably import line as well.	**Lewis (03:51)** new scanner?
Larry (07:10) Oh, you are right. My bad.	**Laura (03:53)** yeah something like that
Larry (07:35) We should also create a scanner	**Lewis (03:55)** I always forget that
Lloyd (07:36) Usually, I call mine “input”	**Laura (03:56)** Yeah me too! Let's see.
Larry (07:43) What is to uhh...	**Lewis (05:46)** new Scanner and then parenthesis system dot in
Lloyd (07:45) Scanner space input	**Laura (05:52)** Oh! new Scanner...
	Lewis (06:04) Like this. That’s on the right
	Laura (06:07) New scanner. Oh ok! Gotcha!

Figure 4. Illustrative excerpts from an interaction between a *Low-Low dyad* in a *Baseline* session on the left and *Think-First* session on the right.

To test the second part of our hypothesis, we to compared the two conditions in terms of enjoyment of the programming activity. There were no significant differences in students’ pair programming enjoyment ratings (Figure 5). In high-performing students paired with high-performing students, we see a decrease in enjoyment from *Baseline* (*mean* = 37.20; *stdev* = 10.16) to the *Think-First* condition (*mean* = 35.06; *stdev* = 9.25) with small effect size (Cohen’s $d = 0.2215$). In high-performing students paired with low-performing students, we see a decrease in enjoyment from *Baseline* (*mean* = 37.50; *stdev* = 8.79) to the *Think-First* condition (*mean* = 31.72; *stdev* = 11.13) with medium effect size (Cohen’s $d = 0.5828$). However, in low-performing students paired with low-performing students, we see an increase in enjoyment from *Baseline* (*mean* = 38.06; *stdev* = 8.31) to the *Think-First* condition (*mean* = 39.83; *stdev* = 9.90) with small effect size (Cohen’s $d = 0.1958$). In low-performing students paired with high-performing students, we see an increase in enjoyment from *Baseline* (*mean* = 35.90; *stdev* = 11.54) to the *Think-First* condition (*mean* = 39.55; *stdev* = 10.23) with small effect size (Cohen’s $d = 0.3296$).

Taken together, the quantitative findings and case studies indicate that the *Think-First* approach provided an opportunity for low-performing students to form goals and to reflect on their previous learning experiences before beginning collaboration. This opportunity improved their collaborative outcomes, with one benefit being that the *Think-First* time decreases how apparent the knowledge gap between students is to each teammate, fostering more equitable collaboration. At the same time, there was no significant decrease in students’ overall enjoyment of pair programming (Figure 5).

Limitations: It is important to acknowledge that this study was not conducted in a controlled environment, but in an actual lab for a CS1 class. There were some students who did not fully complete the lab assignment within the allotted time (two lab hours). These students still completed the posttest and were included in the analysis presented here. Additionally, in this naturalistic classroom environment, some students interacted with classmates from other dyads, and also sought help from the teaching assistant. We did not intervene in these cases. Finally, the overall posttest scores were low (averaging 4.5 out of 10) in both the *Baseline* and *Think-First* conditions. This limitation is due primarily to the fact that students had been scheduled to cover the relevant new material on one-dimensional arrays in their lecture classes before attending this lab, but an unexpected severe weather incident cancelled classes. As a result, students attended their lab before seeing the relevant concepts presented in lecture. Teaching assistants conducted brief mini-lectures in labs to attempt to mitigate the impact, but students still received less exposure to the concepts prior to the lab than was originally intended.

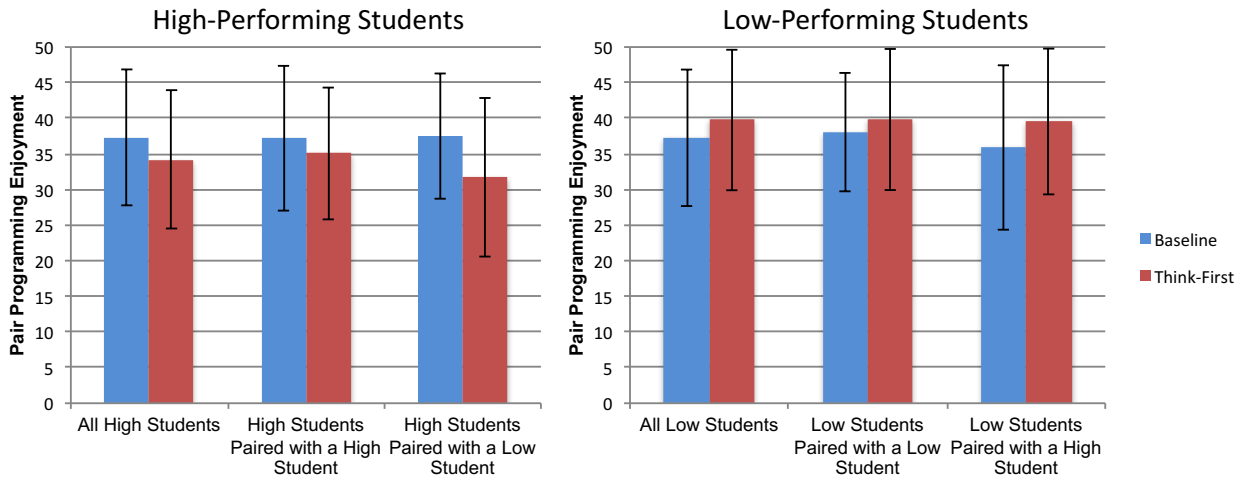

Figure 5. Enjoyment levels across conditions (no significant differences between conditions)

5. Conclusions and Future Work

The CSCL community has long aimed at supporting equity in collaborative problem solving. In this study, we examined the effect of the *Think-First* approach on learning outcomes and collaborative dialogue. We found that it had a significant benefit for learning. Moreover, examination of the dialogues suggests that the individual thinking time enabled students, particularly low-performing students, to formulate their own ideas and build confidence before sharing their ideas with their partners. In the *Think-First* condition, substantive contributions from both partners characterized the dialogues, while in the *Baseline* condition, the low-performing student in a dyad would often become increasingly marginalized over time, turning over control and disengaging from the problem-solving process. The *Think-First* approach holds the benefit of decreasing how apparent the difference in student's previous knowledge is, and may help to foster engagement in both students early in the conversation. As depicted above, the *Think-First* condition seems to change the nature of the students' interactions. In the case of *High-Low* dyads, it seems that the low-performing student was less dependent on the high-performing student, making contributions to both the conversation and the problem-solving task, which promotes healthy collaboration despite one contributor having more incoming knowledge.

There are several important directions for future work. First, while this work has examined one simple type of collaboration structure, there are many types of potential support structures and scaffolds that are important to explore further, such as peer teaching, completing a peer's half-written program, code dividing, and error hunting. Additionally, it is important to more deeply investigate the affective and social components of effective collaboration. Finally, as we move toward supporting diverse learners in problem solving, developing adaptive techniques that consider a rich set of learner characteristics is a promising research direction for computer-supported collaborative learning technologies.

References

Azlina, N. N., & Nik, A. (2010). CETLs: Supporting collaborative activities among students and teachers through the use of Think-Pair-Share techniques. International Journal of Computer Science Issues, 7(5), 18-29.

Beers, P., Boshuizen, H., Kirschner, P. A., & Gijselaers, W. H. (2007). The analysis of negotiation of common ground in CSCL. Learning and Instruction, 17(4), 427-435.

Bevan, J., Werner, L., & McDowell, C. (2002). Guidelines for the use of pair programming in a freshman programming class. In Software Engineering Education and Training, 2002.(CSEE&T 2002). Proceedings. 15th Conference on (pp. 100-107). IEEE.

Chao, J., & Atli, G. (2006, July). Critical personality traits in successful pair programming. In *AGILE 2006 (AGILE'06)* (pp. 5-pp). IEEE.

Cockburn, A., & Williams, L. (2000). The costs and benefits of pair programming. Extreme programming ex*amined*, 223-247.

Davidsen, J., & Ryberg, T. (2015). "This Is the Size of One Meter": Children's Bodily-Material Collaboration and Understanding of Scale around Touchscreens. *Exploring the Material Conditions of Learning: Opportunities and Challenges for Cscl*.

Dybå, T., Arisholm, E., Sjøberg, D. I., Hannay, J. E., & Shull, F. (2007). Are two heads better than one? On the effectiveness of pair programming. *IEEE software*, *24*(6), 12-15.

Fitzgerald, D. (2013). Employing think–pair–share in associate degree nursing curriculum. *Teaching and Learning in Nursing*, *8*(3), 88-90.

Kaddoura, M. (2013). Think pair share: A teaching learning strategy to enhance students' critical thinking. *Educational Research Quarterly*, *36*(4), 3.

Kirschner, P. A., Beers, P., Boshuizen, H., & Gijselaers, W. H. (2008). Coercing shared knowledge in collaborative learning environments. Computers in Human Behavior, 24(2), 403-420.

Kothiyal, A., Majumdar, R., Murthy, S., & Iyer, S. (2013, August). Effect of think-pair-share in a large CS1 class: 83% sustained engagement. In *Proceedings of the ninth annual international ACM conference on International computing education research* (pp. 137-144). ACM.

Kozulin, A., Gindis, B., Ageyev, Vladimir S., Miller, Suzanne M. (Ed.) 2003. Vygotsky's Educational Theory In Cultural Context: Cambridge.

Ludvigsen, S. R., Lund, A., Rasmussen, I., & Säljö, R. (Eds.). (2010). *Learning across sites: New tools, infrastructures and practices*. Routledge.

McCarthey, S. J., & McMahon, S. (1992). From convention to invention: Three approaches to peer interactions during writing. *Interaction in cooperative groups: The theoretical anatomy of group learning*, 17-35.

McDowell, C., Werner, L., Bullock, H., & Fernald, J. (2002). The effects of pair-programming on performance in an introductory programming course. *ACM SIGCSE Bulletin*, *34*(1), 38-42.

Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C., & Balik, S. (2003). Improving the CS1 experience with pair programming. *ACM SIGCSE Bulletin*, *35*(1), 359-362.

Preston, D. (2006). Using collaborative learning research to enhance pair programming pedagogy. *ACM SIGITE Newsletter*, *3*(1), 16-21.

Siburian, T. A. (2013). Improving students achievement on writing descriptive text through think pair share. *IJLLALW*, *3*(03), 30-43.

Stahl, G. (2002). Computer Support for Collaborative Learning: Foundations for a Cscl Community (Cscl 2002 Proceedings). Psychology Press.

Stahl, G. (2006). Group Cognition: Computer Support for Building Collaborative Knowledge (Acting with Technology).

Stahl, G., Koschmann, T., & Suthers, D. (2006). Computer-supported collaborative learning: An historical perspective. Cambridge Handbook of the Learning Sciences, 2006.

Sugiarto, D., & Sumarsono, P. (2014). The Implementation of Think-Pair-Share Model to Improve Students' Ability in Reading Narrative Texts. *International Journal of English and Education*, *3*(3), 206-215.

Vygotsky, L. (1978). Interaction between learning and development. Readings on the development of children, 23(3), 34-41.

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.

Williams, L., Kessler, R. R., Cunningham, W., & Jeffries, R. (2000). Strengthening the case for pair programming. *IEEE software*, 17(4), 19.

Yerigan, T. (2011). Getting active in the classroom. *Journal of College Teaching & Learning (TLC)*, *5*(6).

Acknowledgments

The authors wish to thank the members of the LearnDialogue group at the University of Florida for their helpful input. This work is supported in part by Google through a CS Capacity Research Award and by the National Science Foundation through grant CNS-1622438. Any opinions, findings, conclusions, or recommendations expressed in this report are those of the authors, and do not necessarily represent the official views, opinions, or policy of the National Science Foundation.