Computer Science Education

ISSN: 0899-3408 (Print) 1744-5175 (Online) Journal homepage: http://www.tandfonline.com/loi/ncse20

Pair programming in education: a literature review

Brian Hanks , Sue Fitzgerald , Renée McCauley , Laurie Murphy & Carol Zander

To cite this article: Brian Hanks , Sue Fitzgerald , Renée McCauley , Laurie Murphy & Carol Zander (2011) Pair programming in education: a literature review, Computer Science Education, 21:2, 135-173, DOI: 10.1080/08993408.2011.579808

To link to this article: http://dx.doi.org/10.1080/08993408.2011.579808

Computer Science Education
Vol. 21, No. 2, June 2011, 135–173

Pair programming in education: a literature review

Brian Hanks[a]*, Sue Fitzgerald[b], Renée McCauley[c], Laurie Murphy[d] and Carol Zander[e]

[a]*BFH Educational Consultants, Seattle, WA 98119, USA;* [b]*Department of Information & Computer Science, Metropolitan State University, Saint Paul, MN 55106, USA;* [c]*Department of Computer Science, College of Charleston, Charleston, SC 29424, USA;* [d]*Department of Computer Science & Computer Engineering, Pacific Lutheran University, Tacoma, WA 98447, USA;* [e]*Department of Computing and Software Systems, University of Washington Bothell, Bothell, WA 98011, USA*

(*Received 15 September 2010; final version received 2 April 2011*)

This article provides a review of educational research literature focused on pair programming in the undergraduate computer science curriculum. Research suggests that the benefits of pair programming include increased success rates in introductory courses, increased retention in the major, higher quality software, higher student confidence in solutions, and improvement in learning outcomes. Moreover, there is some evidence that women, in particular, benefit from pair programming. The literature also provides evidence that the transition from paired to solo programming is easy for students. The greatest challenges for paired students appear to concern scheduling and partner compatibility. This review also considers practical issues such as assigning partners, teaching students to work in pairs, and assessing individual contributions, and concludes with a discussion of open research questions.

Keywords: pair programming; collaborative learning

1. Introduction

Pair programming is a technique in which two individuals share a single computer as they work together to develop software. Although pair programming has been used in industry since the 1970s (Jensen, 2003), it has become much more popular in education in the last 10 years as part of the agile software development movement (Beck, 2000). Its use in educational settings has also increased significantly since first being used at the University of Utah in the late 1990s (Williams, 2000), with reported usage in the US (for example, Braught, Eby, & Wahls, 2008; McDowell, Werner, Bullock, & Fernald, 2003; Williams, & Kessler, 2001), the United Kingdom (Chaparro, Yuksel, Romero, & Bryant, 2005; Thomas,

*Corresponding author. Email: hanks.brian@gmail.com

ISSN 0899-3408 print/ISSN 1744-5175 online

DOI: 10.1080/08993408.2011.579808
http://www.informaworld.com

Ratcliffe, & Robertson, 2003), Germany (Müller & Tichy, 2001), New Zealand (Mendes, Al-Fakhri, & Luxton-Reilly, 2006), India (Kuppuswami & Vivekanandan, 2004), and Thailand (Phongpaibul & Boehm, 2006).

This article provides a review of the pair programming research literature and is focused on the study of pair programming in the undergraduate curriculum. A majority of these studies focus on introductory programming, although research on pair programming in upper-level and graduate courses, and some from industry as it relates to education, is also reported.

Section 2 summarizes research findings about the benefits of pair programming in higher education. Section 3 addresses the challenges of scheduling and partner compatibility, and the transition to solo programming. Section 4 addresses the practical matters of introducing paired programming in classroom settings. Section 5 closes with a summary of open research questions. In the appendix, a table maps papers to the sections where they are discussed.

2. Educational benefits of pair programming

In educational settings, pair programming is frequently used in introductory programming courses as there is notable evidence that its use helps students learn to program. However, it has also been used in second-year courses, in upper division courses such as software engineering or object-oriented programming, and even at the graduate level. In this section, research findings about the benefits of pair programming in an educational setting are presented.

2.1. Success in first- and second-year programming courses

Learning to program is difficult for many students. Failure and withdrawal rates in introductory programming courses of 33% or greater are not uncommon (Bennedsen & Caspersen, 2007). Studies of pair programming in introductory programming courses offer compelling evidence that it can help reduce these high failure rates.

One of the most extensive studies of pair programming was conducted by McDowell, Werner, Bullock, and Fernald (2003, 2006) in 2000 and 2001. They studied the impact of pair programming on student performance in four sections of an introductory programming course (CS1) at the University of California, Santa Cruz. Students in three of the sections worked in pairs ($n = 404$), while students in the fourth section worked alone ($n = 148$). The paired students worked on all of their programming assignments (both closed labs and homework) with their partner. There was no evidence for a difference between the groups in

terms of grade point averages (GPA) and SAT exam scores. (The SAT exam is typically taken by high school students in the US to assess their preparation for college).

Paired students were significantly more likely than non-paired students to take the final exam (90.8% vs. 80.4%, $\chi^2(1) = 11.21$, $p < 0.001$) and to pass the course (72.3% vs. 62.8%, $\chi^2(1) = 4.57$, $p < 0.05$). Although a larger percentage of paired students took the final exam, both groups earned similar scores (pairs: 75.2%, solos: 74.4%). As a result, a larger proportion of the paired students demonstrated sufficient mastery of the course material to pass the final exam and the course. This suggests that paired students are as able to apply their learning independently as students who work alone.

Nagappan et al.'s (2003) study of pair programming found that final course grades for students engaged in pair programming were equal to or better than those of solo programmers. Students in CS1 were arbitrarily assigned to closed lab sections where they were either partnered or required to work individually. Partners were assigned randomly and changed every 2–3 weeks. The experiment ran for three semesters and included 661 students (380 paired; 281 solo). Data analysis was limited to first- and second-year students who were taking the course for a grade.

Students were also given programming assignments which were to be done outside the closed lab setting. There was some variation in expectations for pairing vs. individual work on these assignments as the experiment progressed. In the first semester, all students enrolled in the class were permitted to pair on their programming assignments if they wished. In the second semester of the experiment, pairing on outside programming assignments was limited to those students who performed well on their tests (scores of 70% or higher). In the third semester, students were required to work on their programming projects with their lab partners. As a consequence, data analysis was performed separately for each semester. Overall, Nagappan et al. (2003) found that course grades for paired students were equal to or better than the course grades for those students who worked individually in the labs.

In a 2-year study at a small liberal arts college, Braught et al. (2008) investigated the impact of pair programming in CS1. In their study, students in four sections of CS1 programmed by themselves, while students in four other sections worked in pairs during supervised lab sessions. All other works (including programming assignments) were completed individually. Students in both groups had similar SAT scores.

The students' programming abilities were assessed by individual lab practica in which students had 2 h to work on programming problems. For students with lower SAT scores (less than 1265), the students in the paired sections of the course received higher lab practica scores than those in the solo sections. There was no statistically significant difference in lab

practica scores between the paired and solo sections for the students with higher SAT scores; these students tended to receive high lab practica scores regardless of their section.

This research also showed that, for a given SAT score, students who paired were more likely to pass the course (with a grade of C or better) than those who worked alone. The odds of paired students passing the course were approximately three times that of solo students. These results suggest that pair programming may be particularly helpful for weaker students.

Somervell (2006) also investigated pair programming in CS1. In his study, 16 students in one section of CS1 paired, while 20 students in a second section worked alone. The instructor, exams, assignments, and lecture material were the same for both sections. Somervell hypothesized that students in the paired section would outperform students in the solo section in terms of grades on programming assignments, exams, and in the course in general. However, none of his hypotheses were supported by the data, and he found no evidence that pairing provided any benefits for students. Students in both sections performed equivalently on all measures.

Mendes, Al-Fakhri, and Luxton-Reilly (2005) investigated the impact of pair programming in a second-year software design and construction course in New Zealand. In this study, 114 students paired and 186 students worked alone on their closed laboratory exercises. These exercises were for practice only, and were not included as part of the course grade. All other work in the course was done individually.

They found that the students who worked in pairs for lab exercises passed the course (with a grade of C or better) at a higher rate than the students who worked alone. They obtained similar results when they replicated their experiment the following year (Mendes et al., 2006). In their follow-on study, they also found that paired students earned higher grades on their individual work, including programming assignments, tests, and the final exam.

Taken together, these studies provide some evidence that pair programming leads to increased student success in introductory and intermediate programming courses.

2.2. Success in later courses

Educators may be concerned that students who pair in their first programming course might not succeed in later courses where they have to work alone. It appears likely that this concern is unfounded, because research evidence thus far has shown no detrimental effect of pairing on the success of students required to work alone in later courses.

McDowell et al. (2006) tracked their study participants for a year after they had taken their introductory programming course. Of the students

who took the second programming course, those who pair programmed in the introductory course were more likely to pass the second course on their first attempt (65.5% pairs, 40.0% solos), even though all students were required to work alone in the second course. It appears that, by pairing in the first course, students were able to build a foundation that led to later success when they had to work alone. It also seems to refute the notion that pair programming results in weak students passing the first programming course by over-relying on their partners, only to fail later when they have to work alone.

Nagappan et al. (2003) also examined the performance of CS2 students who paired in CS1 in order to allay concerns about weaker students getting a ''free ride'' in CS1. The CS2 course had stringent requirements for solo programming on all assignments. They found mixed results. In one term, formerly paired students performed much better in CS2 than those who were not given the opportunity to pair in closed labs in CS1 (69.70% of the formerly paired students passed CS2 with As or Bs compared to 44.82% of the formerly solo students). In the following term, solo students performed better than formerly paired students (85.25% of the formerly solo students passed the course with As or Bs compared to 71.43% of the formerly paired students). Both results were statistically significant. Nagappan et al. (2003) went on to analyze those students whose grades in CS2 dropped below their CS1 grades. For both terms, grades dropped for a greater percentage of formerly solo students than formerly paired students (21.42% of paired students' grades dropped in CS2 vs. 46.15% of the solo students in the first term; 26% of paired students' grades dropped in CS2 vs. 30% of the solo students in the second term). They conclude, ''pair programming is not detrimental to a student's performance in future programming courses done in solo'' (Nagappan et al., 2003, p. 194).

Jacobson and Schaefer (2008) found no evidence that students who pair in their first programming course struggle when asked to program individually in subsequent courses. Survey results and instructor conversations indicated that students in following courses had no difficulty switching from pair to individual programming.

2.3. Retention

Retention in computing-related majors is a significant concern, particularly when enrollments decline. Allowing students to pair program in their introductory course appears to have a positive effect on retention.

Carver, Henderson, He, Hodges, and Reese (2007) conducted a 2-year study of pair programming in CS1 and its impact on student retention in computing-related majors. To measure retention, they recorded the major of all students who were freshman when they took CS1, and then looked

at the major of these students 1 year later. In one section of the course, 75 freshmen pair programmed on their closed laboratory exercises, while 79 freshmen in three other sections did not pair. All other works in these courses were completed by students working alone. For students who were computing majors when they took the CS1 course, 85.4% (41 of 48) of those in the paired section were still majors 1 year later, while only 64.3% (36 of 56) of those who worked alone in CS1 were still majors. This difference in retention rates was statistically significant ($p = 0.014$). However, pair programming did not attract non-majors, as no students who were not majors when they took CS1 were majors 1 year later. On the other hand, using pair programming did not appear to discourage non-majors, as none of the soloing non-majors became CS majors either.

Students who paired in McDowell et al.'s study (2006) were much more likely to take the second programming course. Of the students who indicated that they intended to pursue a computing-related major when they were enrolled in CS1, and passed CS1 with a grade of C or better, significantly more of those who paired attempted the second programming course within 1 year (84.9% of paired students vs. 66.7% of solo students). These students were also much more likely to persist in a computing-related major. For those students who indicated a desire to major in computer science when they took CS1, the pairing students were significantly more likely to have declared a computing-related major 1 year later: 70.8% of paired students compared with 42.2% of solo students ($\chi^2(1) = 12.18$, $p < 0.001$, McDowell et al., 2003).

2.4. Quality of programs produced by pairs

The ultimate goal of programming courses is to teach students how to develop quality software. Research studies have shown that pairs produce higher quality code than solo programmers, although this does not necessarily suggest that the students who paired learned more as a consequence. In these studies, software quality has been measured in many different ways.

Williams and Kessler (2001) measured quality according to the number of test cases the software passed. They found that programs developed by pairs passed significantly more ($p < 0.01$) of the instructor's test cases than those developed by solo programmers.

McDowell et al. (2003) also found that students who paired produced higher quality programs than those who worked alone. They measured quality (based on functionality and readability) as the normalized average score on programming assignments, and found that paired students earned an average score of 86.6% while unpaired students received scores that averaged 68.1%. This difference was statistically significant ($F(1,482) = 77.42$, $p < 0.001$).

Although pairs in the McDowell et al.'s (2003) study produced higher quality programs than students who worked alone, the two groups did not work on the same assignments. Programs given to the paired and unpaired students were intended to be of similar complexity, yet it is possible that the unpaired students' assignments were more challenging, and as a result they did not do as well. In an effort to address this question, Hanks, McDowell, Draper, and Krnjajic (2004) conducted a follow-on study to investigate the differences in the quality of programs produced by paired and unpaired students. In this study, students in a paired section of CS1 were given the same programming assignments as the students in the solo section of the McDowell et al.'s (2003) study. Hanks et al. (2004) compared the programs produced by the paired and unpaired students using both objective and subjective measures. Objective measures included program length, number of features correctly implemented, and cyclomatic complexity (McCabe, 1976). Subjective measures included the use of appropriate variable names, method organization, and programming style.

Hanks et al. (2004) found that pairs were able to correctly implement more of the required features in their programs than unpaired students. For other measures, their results were inconsistent. On some programming assignments, pairs produced programs that were less complex and better designed, while on other assignments the opposite was the case.

However, Hanks et al. (2004) reported one unexpected, but important, result. Paired students were much more likely to turn in solutions to the programming assignments, and these solutions were more likely to compile without error than solutions turned in by students who worked alone. These greater submission and error-free compilation rates suggest that paired students are able to overcome problems that frustrate solo students, who may give up.

Phongpaibul and Boehm (2006) compared the quality of programs developed by student teams in a junior-level software engineering class in Thailand. They studied two conditions: teams in which students pair programmed, and teams in which students worked alone and used software inspections (Ackerman, Buchwald, & Lewski, 1989) to find defects. These conditions were randomly assigned to the student teams. The pair development groups' projects had fewer problems than the projects developed by the inspection groups. There were statistically significant differences in the number of major problems and the total number of problems between the two groups. This suggests that teams of pairing students produce higher quality work than teams of students working independently, even when those teams use software inspection, which has been shown to be a highly effective technique for defect reduction (Ackerman et al., 1989).

Bipp, Lepper, and Schmedding (2008) examined the impact of pair programming on teamwork. In their study, 63 second-year students were

randomly assigned into teams of seven or eight people. The teams were then randomly assigned to one of two working conditions: the team members programmed individually, or they pair programmed within their team. They found that the programs developed by the teams in which the students worked in pairs were less complex in terms of coupling and cohesion. The code produced by the paired teams was also judged by external experts to be "a little bit better" and was more readable and understandable.

Students also report that pairing leads to fewer programming errors. In a study of CS2 students, DeClue (2003) reported that students believed that they made fewer errors when they programmed in pairs, and that the errors they did make were found sooner. Similarly, Melnik and Maurer (2002) found that 84% of undergraduate and graduate students, in three separate classes at two Canadian institutions, agreed with the statement "I believe that pair programming improves software development quality (better design, code easier to understand)".

Improved quality is also seen as a benefit by professional programmers. In a survey of 106 software developers at Microsoft with pair programming experience, the most frequently cited benefit of pairing was that the resulting code had fewer bugs (Begel & Nagappan, 2008). Another frequently cited benefit in this study was that the partners could learn from each other, which is also a benefit in educational settings.

It is important for computing instructors to remember that the goal of programming courses is for students to learn how to program, and that the various measures of program quality used in these studies may not be strong indicators of student learning.

2.5. Confidence in solutions

Much of the research investigating student confidence has looked at the influence of perceived self-efficacy on programming success, with mixed results (e.g, Cantwell Wilson, 2002; Ramalingam, LaBelle, & Wiedenbeck, 2004; Wiedenbeck, 2005). Research into pair programming, however, has looked more specifically at the influence of pair programming on students' confidence in the correctness of their solutions. Although studies that directly linked pair programming with objective measures of perceived self-efficacy were not found, Bandura's (1977) extensive research on self-efficacy suggests that observing peers while they successfully complete challenging tasks and social persuasion, both natural components of pair-programming, can improve self-efficacy.

Many studies have shown that students who pair are more confident than those who program alone, although some students are ambivalent about the relationship between pairing and confidence. In the McDowell

et al.'s (2003) study discussed earlier, students answered the following question when they turned in their programming assignments:

> On a scale from 0 (not at all confident) to 100 (very confident), how confident are you in your solution to this assignment?

The paired students were significantly more confident in their work than the students who worked alone (89.4 vs. 71.2, $F\ (1,482) = 99.38$, $p < 0.001$).

In the Hanks et al.'s (2004) study which compared the quality of student programs produced by solos and pairs, they asked their subjects the same question, and found confirming evidence that pairs are more confident in their work. Students who worked in pairs indicated a confidence level of 81.6 compared with a confidence level of 72.7 for unpaired students. The probability that paired students actually were more confident in their work was 99.9%.

In their study of eight sections of CS1 in which students in four sections paired for laboratory assignments, Braught, Wahls, and Eby (2011) asked students to indicate their level of agreement with the statement, ''When I submitted my laboratory assignments I was confident that they were correct,'' and found that students who paired were more confident in their solutions on their lab practica than students who worked alone ($F\ (1, 136) = 6.35$, $p = 0.013$). The paired students were also more confident in the completeness of their testing ($F\ (1, 137) = 8.87$, $p = 0.003$). Braught et al. (2011) asked similar questions about homework assignments and exams, and found no difference between the paired and solo students. For this study, pair programming was used only for lab assignments; all other works in the class were done by students working alone.

End-of-term survey results from a large study of CS1 students (VanDeGrift, 2004) also suggests that pairing increases student confidence. Students in the course generally agreed with the statement, ''I had more confidence in my solutions to the pair programming projects than the individual homework assignments in this course''.

DeClue (2003) asked students who paired in a CS2 course to answer the question, ''Did having a partner increase your confidence that your code was more reliable (i.e. had fewer bugs, wouldn't crash as easily)?''. The question was asked three times during the semester (when students changed partners). The response was ''overwhelmingly positive'' (DeClue, 2003, p. 52), showing that students believed having a partner gave them more confidence in the reliability of their programs.

In a graduate level course on extreme programming, 75% of students indicated that one of the advantages of pair programming was that they had more confidence in their solutions of their project assignments (Müller & Tichy, 2001).

In one of the earliest studies of pair programming in academia, more than 80% of students who pair programmed in two upper division software development courses indicated they had greater confidence in their work when they pair programmed (Williams, 2000). Two years earlier, Nosek (1998) reported similar findings with programmers working in industry.

Qualitatively, students' views regarding the relationship between pair programming and confidence may be less consistent. Simon and Hanks (2008) interviewed eleven students who paired in their first programming course but went on to program individually in their second programming course. Although two of the subjects reported that transitioning to solo programming lowered their confidence in their programming ability, five reported that their confidence increased as a result of solving the problems by themselves. Although this study looked at confidence in programming ability instead of solutions, it suggests that it may be important for students to program individually at some time to build their confidence.

2.6. Attitudes toward pairing

In general, the literature shows that students enjoy pair programming more than solo programming, with few exceptions. Students claim that pair programming is fun, enjoyable, and useful for a wide variety of tasks.

Nosek (1998) found, with statistical significance, that programmers working in pairs enjoyed the experience more than those working alone. Similarly, Williams (2000) found that 95% of programmers enjoyed their programming work more when they worked collaboratively rather than individually.

Chaparro et al. (2005) conducted an exploratory study of pair programming in a post-graduate, object-oriented programming course. In this course, students were required to pair in three laboratory sessions, which each focused on different activities: program comprehension, debugging, and refactoring. Fifty-eight students (out of 80 in the course) volunteered to participate as subjects, but there is no indication of the subjects' experience level or other background information. Experimental data were collected from participant observation, questionnaires, semi-structured interviews, and field notes. By triangulating the data collected via these different means, Chaparro et al. (2005) were able to provide stronger support for their results.

In data from their observations and field notes, Chaparro et al. (2005) determined that students enjoy the collaborative experience of pairing, as they "celebrated every achievement while trying to solve the exercise." However, they also noted that higher skilled students sometimes became frustrated when things were going wrong. This was not a problem for lower skilled students, as for them doing something wrong was not

unexpected, and they were happy to have someone else to work with to overcome their problems.

Chaparro et al. (2005) found that the majority of students agreed with the statement, "I enjoyed doing pair programming today." However, the percentage of students who agreed differed based on the task that was being done: 84% of students enjoyed pair programming when working on program comprehension tasks, 78% enjoyed it for refactoring tasks, but only 58% found it enjoyable when debugging.

In contrast, end-of-term survey results from VanDeGrift (2004) found that CS1 students not only enjoyed working together on pair projects, but also felt pairing helped them debug more effectively. Students generally agreed with the statement, "I was more efficient in debugging my code while working with a partner on the projects versus working individually on the homework assignments".

McDowell et al. (2003) found that paired students enjoyed working on their programming assignments more than students who worked alone. When students turned in their programming assignments, they responded to the following question: "How much did you enjoy working on this programming assignment? (1 = not at all, 7 = very much)." Paired students reported a mean enjoyment level of 5.15, which was significantly greater than the mean level of 4.69 reported by the non-paired students ($F(1, 482) = 9.00$, $p < 0.005$).

Thomas, Ratcliffe, and Robertson (2003) paired first-year students with prior programming experience for two assignments. Using self-perception of programming confidence, students rated themselves on a scale from 1 to 9. A low number meant they did not like programming, they did not think they were good at it, and they had trouble writing new programs (the 'Code-a-Phobes'). A high number meant they had no trouble completing programming tasks, loved to program, and anticipated no difficulties (the 'Code Warriers'). Students were categorized into three groups based on their self-perception rating: 1–3 (17 students), 4–6 (34 students), and 7–9 (13 students).

For the first assignment, students in the high and low self-perception groups were paired with opposites, while middle group students were paired with each other. Overall, 66% of the students enjoyed the experience. However, the most self-confident students reported enjoyment only 53% of the time.

For the second assignment, students were given partners with similar confidence levels. The pairs of high confidence 'Code Warrier' programmers reported higher levels of satisfaction than they had when paired with less confident partners (58% enjoyed the second assignment vs. 53% in the first assignment). However, they still enjoyed the experience less than the rest of the participants (58% vs. 64% overall). Overall, 44% reported liking the second experience more than the first (and 38% reported liking

it as much), implying that matching confidence levels leads to higher satisfaction. Of particular interest, a higher percentage, 60%, of the less confident 'Code-a-Phobe' group said they enjoyed the second experience more than first experience where they were paired with highly confident partners. Although these results suggest that less confident students enjoy pair programming more, none of the differences were statistically significant.

Hanks (2006) examined student attitudes toward pair programming, by surveying 115 students in three CS1 classes where pair programming was used. He asked students to indicate their level of agreement with the statements, ''I like pair programming,'' and ''I had more fun in this class because I pair programmed.'' The mean response levels over all classes to these questions were highly positive, and significantly differed from a neutral response ($p < 0.001$).

Hanks (2006) also examined the relationship between student confidence and attitudes toward pair programming. In his study, the most confident students also enjoyed pair programming the most. Although this finding seems to contradict that of Thomas et al. (2003), different measures of confidence were used in the two studies. As Thomas et al. (2003) note, their measure of confidence was conflated with skill level, while Hanks (2006) used students' self-reported confidence in their programming assignment solutions as his measure.

Nagappan et al. (2003) surveyed their paired CS1 students to determine the students' attitude toward pair programming. They asked, ''If you are in a paired section this semester, will you choose a paired section course in the next semester, given there is a paired section?'' In the first term of the experiment, 59.9% of the paired programmers said they would choose pair programming again. In the second round of the experiment, 64.7% of the paired CS1 students said they would choose pair programming again. Nagappan et al. (2003) inferred that ''students in paired labs have a positive attitude toward collaborative programming settings'' (p. 196), even though more than a third of the students each term would not have chosen pair programming again.

2.7. Impact and perceived impact on learning

Reports on measurable learning outcomes show that pair programming is beneficial. Moreover, several studies have examined students' impressions of the impact of pairing on learning, and have found that students believe pair programming leads to improved learning. Observations of student interactions suggest that collaborative and explanatory activities lead to deeper-level thinking. On the other hand, interviews of a small sample of students reveal that during pairing some students did not fully understand what their partners were doing.

In a series of experiments involving 214 students (as 58 pairs and 98 solos), Kuppuswami and Vivekanandan (2004) had students create solutions to small programming problems. Initially, the students developed design documents for their programs, which were corrected by a laboratory instructor. The students then spent as much time as necessary to develop a correct program (i.e. a program that passed all test cases) – the amount of time required ranged from about 75 to 215 min. Immediately after completing the program, the students individually took a paper-based exam which assessed programming skills and knowledge gained in the exercise.

Kuppuswami and Vivekanandan (2004) found that the students who paired in these experiments performed better on the individually-taken post-tests than the students who worked by themselves. The paired students also received higher marks on their program designs. This suggests that working in pairs leads to improved learning.

In a large study of CS1 students (VanDeGrift, 2004), positive results were obtained from Likert survey responses to both ''I learned about concepts covered in the course by working with a partner on projects.'' and ''I gained more understanding of concepts in the course by explaining them to my project partner''.

In their study of second-year students working in teams, Bipp et al. (2008) required pair programming in some teams but not in others. They found that students who worked individually in project teams indicated greater agreement with the statements, ''The work load was not equally distributed in our group'' and, ''I barely know some parts of the project'' than students on paired teams. These differences in response levels were statistically significant. This finding suggests that using pairing within teams is an effective way to get all members to contribute equally, and improves learning because the students have greater familiarity with all aspects of their projects. It appears that pair programming may be an effective way to integrate less experienced or less confident students into project teams as equal contributors.

As part of their 2-year study of pair programming in CS1, Carver et al. (2007) asked students to indicate their level of agreement with the statement, ''I learned more working with a partner on this programming assignment than I would have without a partner.'' They found that the students strongly agreed with this statement. Although this question only assesses student impressions of learning, it suggests that students believe that one of the benefits of pair programming is the opportunity to learn from a partner. As noted earlier, professional programmers express the same view (Begel & Nagappan, 2008).

In his study of student attitudes toward pair programming, Hanks (2006) asked students to indicate their level of agreement with the statement, ''I learned more in this class because I pair programmed,''

using a 7-point scale where "1" indicated strong disagreement and "7" indicated strong agreement. The mean response level for all students was 5.07 (which differed significantly from the neutral response of 4 ($p < 0.001$)), indicating that students believe pair programming helps them learn.

Chaparro et al. (2005) also asked students to provide a measure of their impression of learning due to pair programming. After each of the three paired laboratory exercises in their study, students indicated their level of agreement with the statement, "I think I've learned more this time working in pairs than others that I've worked on my own." The students tended to agree with this statement, which suggests that they believe that pair programming helps them learn. There was also an indication that there is a relationship between a student's skill level and his or her perception of learning, as less-skilled students frequently reported that they learned more by pairing.

There was also a significant interaction between perceived learning and task type. On the program comprehension task, 55% of the subjects agreed with the statement, while only 4% disagreed. The percentages were similar for the refactoring task, with 65% agreeing and 6% disagreeing. However, on the debugging task, 48% of the students agreed that they learned more by pairing, but 22% disagreed. This finding suggests that pair programming may not be as useful for learning about debugging as it is for learning other types of programming tasks.

In Thomas et al. (2003), where students reporting extreme opposite confidence levels were paired and students with middle-ranging confidence were paired with each other, 66% of the students thought pair programming helped them produce a better solution. But the most self-confident students thought pair programming led to a better solution only 47% of the time. For a second assignment, students were all paired from the same groups. In the high confidence group, 67% said the pairing led to a better product (compared to 65% overall).

In their study of computer science graduate students enrolled in an extreme programming course, Müller and Tichy (2001) asked the students to identify the advantages of pair programming. Every student listed the ability to learn from their partner as an advantage.

Cao and Xu (2005) observed student pair programmers to identify the types of interactions that they used. In their study, the student participants had taken at least one prior programming course, and also had IT work experience. The students in their pairs commonly engaged in the following activities: asking for advice, requesting and giving explanations, critiquing each other's approach, and summarizing just completed tasks. They argue that these collaborative and explanatory activities account for pair programming's educational benefits, because they promote deeper-level thinking.

Students who paired during a 6 week project in a CS2 course felt that pair programming allowed them to think of more ideas about possible programming solutions (DeClue, 2003). This theme was echoed by the students interviewed by Simon and Hanks (2008) (who paired in their first programming class but worked alone in their second), who said that they explored more ideas or potential solutions when they paired.

On the other hand, the students interviewed by Simon and Hanks (2008) also stated that they understood their code better when they worked individually. Their comments revealed that during pairing they sometimes did not fully understand what their partners were doing. At the same time, the students recognized the power of applying two minds to solving a problem.

As Kuppuswami and Vivekanandan (2004) notes, students who pair are able to answer more of their own questions, instead of asking a laboratory instructor for assistance. Nagappan et al. (2003) collected qualitative data from focus groups and observations of closed labs. The lab instructors in solo programming sections were often overwhelmed by students with basic syntax questions and were unable to assist all students who had questions in the time allocated. On the other hand, paired students were able to answer most of these basic syntax questions by helping each other. They asked more sophisticated questions about their algorithms when they did interact with the lab instructor. Hanks (2008b) also found that paired students asked fewer questions in closed labs, but in his study the proportion of basic syntax questions was the same for paired and solo students.

2.8. Social aspects of pairing

Pair programming provides students with an opportunity to meet their peers, which can be especially important for firs-year students. The peer relationships established by pairing in the introductory programming course often lead to support groups that help students succeed in later courses (Jacobson and Schaefer, 2008). Jacobson and Schaefer (2008) also note that pairing helps students learn to work well with others, a valuable social skill. Somervell (2006) notes that pairing gives students valuable teamwork experience.

Students surveyed by Cliburn (2003) confirm this, as 92% of them agreed that the experience of pair programming made them better at working with others. Cliburn (2003) argues that this result makes, ''the strongest case for pair programming in the introductory programming course'' (p. 27).

Simon and Hanks (2008) also reported on the social aspects of pair programming. They interviewed eleven students who initially learned to program as pairs, then solo programmed in their second course. These

students expressed an increased feeling of pride and accomplishment when they completed their programming assignments individually although they found pair programming less frustrating. They also enjoyed the social aspects of pair programming and found it a good way to meet their fellow students. Simon and Hanks (2008) speculate that this formalized social relationship which breaks down isolation may be a cause of the increased retention often reported as a result of pairing.

2.9. Impact on women

Although pair programming provides many benefits for students, there is some evidence that it benefits women to a greater degree than men. This suggests that pair programming may help to reduce some aspects of the gender gap in computing.

As discussed by Margolis and Fisher (2002), female students in programming courses are frequently less confident than men, even when their actual level of competence is the same. This leads them to conclude that they do not "belong" in computing, with the result that they leave at higher rates than men. Because fewer women than men attempt computing-related majors in the first place, this higher drop rate results in a greater gender gap in computing degrees than would otherwise be the case.

In the study of pair programming in CS1 conducted by Werner, Hanks, and McDowell (2004), paired women were more likely than those who worked alone to take the final exam (88.1% vs. 79.5%). Although a greater percentage of paired women took the exam, the pass rates for both groups were similar, leading to greater overall course success for paired women. While the differences in completion and success rates for paired and solo women were not statistically significant, Werner et al. (2004) argues that more than 8% increase for paired women is of practical significance.

Among the students who passed CS1 (with a grade of C or better), Werner et al. (2004) found that a significantly higher percentage of paired students attempted the second programming course: 76.7% of the paired students took the second course, while only 62.2% of the unpaired students did ($n = 237$, $\chi^2(1) = 6.17$, $p < 0.05$). Although pairing was related to an 18% increase in attempt rates for both paired men and women, this difference was significant for men ($n = 186$, 88.0% vs. 69.4%, χ^2 $(1) = 7.60$, $p < 0.01$), but not for women ($n = 51$, 73.8% vs. 55.6%, $\chi^2(1) = 1.19$, $p = 0.27$). The authors suggest this lack of significance may be due to the small number of women in the study.

For women who indicated on the first day of CS1 that they intended to pursue a computing-related major, there was a greater likelihood that they would have actually declared such a major 1 year later if they were

paired. For paired women, 55.5% were computing majors 1 year later, compared with only 22.2% of the women who had worked alone ($\chi^2(1) = 4.14$, $p < 0.05$) (Werner et al., 2004).

In the area of confidence, Werner et al. (2004) also found that pair programming had a significant impact. Paired women were more confident than solo women (86.8 vs. 63.0, $p < 0.001$), and paired men were more confident than solo men (90.3 vs. 74.6, $p < 0.001$). Moreover, pair programming had a greater impact on the confidence of women than that of men. Unpaired men indicated a confidence level 11.6 points higher than unpaired women, but for paired students the difference in confidence levels between women and men was only 3.5 points. This suggests that pair programming "may have a visible, positive impact on the gender gap" (Werner et al., 2004, p. 50).

Braught et al. (2011) asked students to indicate their level of frustration while working on their laboratory assignments, using a 5-point scale ranging from very high to very low. In the courses taught by one of the two instructors, female students who worked alone were more frustrated than those who worked in pairs ($t(133) = 2.28$, $p = 0.024$). Solo females were also more frustrated than solo males (($t(133) = 2.82$, $p = 0.006$), but there was no difference in frustration between paired females and paired males. No differences were found between the men and women in the second instructor's courses. The authors suggest that pair programming may help reduce the amount of frustration experienced by female programmers.

In his survey of CS1 students who paired, Hanks (2006) found that female students ($n = 28$) had more positive impressions of pair programming than men ($n = 87$). Women students indicated greater agreement than men with all questions ("I like pair programming," "I would like to pair program again in another class or in my job," "I learned more in this class because I pair programmed," and "I had more fun in this class because I pair programmed"), although the differences were not statistically significant.

Carver et al. (2007) measured the number of computer science majors in CS1 who were still majors 1 year later, and found that two-thirds of the female students who pair programmed were still majors 1 year later, while only one-third of the females from the solo class were. Although encouraging, this difference was not statistically significant due to the small number of women in the study (nine in the paired class, six in the solo class).

In a study that coupled pair programming with written reports, VanDeGrift (2004) found that anonymous ratings of enjoyment, solution confidence, learning of concepts, and debugging efficiency were nearly equal between male and female pair programmers. However, women's impressions of the written reports as a means of increasing confidence,

technical understanding, and reflection were higher than men's, although no statistical significance for these results is presented.

3. Challenges

While the benefits of pair programming are notable, there are some challenges. Students' ability to schedule time to work with partners is hindered by busy personal, school, and work lives. Partner compatibility is an issue leading instructors to examine personality traits and compatibility. Additionally, faculty members are concerned that students will not learn to program individually.

3.1. Scheduling

One of the most significant challenges faced by students who pair appears to be scheduling time to work together. Students are often busy and have conflicting class schedules, and many of them also have work and family obligations that make it difficult to find times to meet their partners.

Scheduling may be especially challenging when students do not live on or near campus. Pair programmers in a large CS1 course at a primarily commuter university reported that finding time to meet their partner posed the biggest obstacle in the course (VanDeGrift, 2004). It should be noted that unlike many introductory courses which use pair programming, this course did not include a closed laboratory component.

Some of the students in the courses examined by Melnik and Maurer (2002) found that work and other commitments interfered with their ability to schedule pair programming sessions. This limited the number of hours per week that they could work together, with the result that these pairs were not as successful as those whose schedules were more flexible.

DeClue (2003) also reported scheduling as an issue for his students, with students' comments including "It was difficult to find time to get together," and "Finding time to meet out of class is always difficult." Some pairs in Cao and Xu (2005) also had scheduling difficulties, particularly because they were working on a complex project that required them to meet frequently on their own time. Bevan, Werner and McDowell (2002) found the scheduling difficulties were one of the most common problems reported by paired students.

Scheduling problems were also frequently mentioned by the students who were interviewed by Simon and Hanks (2008). These students, who had experience in both paired and solo programming, expressed the view that scheduling and working styles were sometimes negatively impacted by pairing. The students commented favorably about the freedom of setting their own schedules when solo programming. Some preferred to start projects early and were discouraged from doing so when pairing.

Others, more prone to procrastination, found pairing more helpful from a time management standpoint.

One way to avoid scheduling problems is to require pair programming in closed labs only (where attendance is required). It appears that students receive many of the benefits associated with pair programming even when it is only used in closed labs (Braught et al., 2008; Carver et al., 2007; Chaparro et al., 2005).

Interestingly, scheduling pair programming sessions is also seen as a challenge in industry (Begel & Nagappan, 2008), because of time conflicts and because it reduces employees' choice of working hours.

3.2. Partner compatibility

One concern that educators may have with pair programming is that some pairs will be incompatible, which will interfere with student learning and lead to an increased workload for the teaching staff who need to address the problems associated with these pairs (Jacobson & Schaefer, 2008).

In general, this concern appears to be unfounded as several studies showed that the majority of students get along well with their partners. Carver et al. (2007) used a 5-point scale to ask students to indicate how well they got along with their partner. The mean response of 4.44 differed significantly from the neutral response of 3, indicating that the students overwhelmingly felt that they were compatible with their partners.

Hanks (2005) asked students who paired in their CS1 course to indicate their level of agreement with the statement, ''My partner and I worked well together'' using a 7-point Likert scale, where 1 indicated ''strong disagreement'' and 7 indicated ''strong agreement''. A significant majority (66.5%) of the students' responses were either 6 or 7. Only 14.7% of students expressed some level of disagreement with this statement.

In a CS1 class at UC Irvine, students were allowed to choose their own partners. Fewer than 5% of the pairs had compatibility problems, based on reports made by either partner or by a teaching assistant (Jacobson & Schaefer, 2008).

Bevan, Werner, and McDowell (2002) found that one of the most common stresses reported by paired students was a significant experience disparity between the partners. The more experienced students were frequently impatient with their partners and several students described having a less adept partner as a ''waste of their time'' and would write the program alone. Although other students complained of incompatibilities, fewer than 2% had issues that were serious enough that they eventually re-paired. Overall, fewer than 5% of all students reported significant scheduling or reliability conflicts. However, logs students kept revealed some unreported conflicts.

In their observational study of student pairs, Cao and Xu (2005) noted that pairs in which the partners had very different skill levels did not work as effectively as pairs where the partners had similar skill levels. For example, the low-skilled partner would simply agree with any approach taken by the higher-skilled partner, and the higher-skilled partner would tend to ignore suggestions made by the lower-skilled partner. In these mismatched pairs, the highly competent partner did not like pair programming, although the weaker partner did.

Sennett and Sherriff (2010) confirmed the findings of Katira et al. (2004) and Williams, Layman, Osborne, and Katira (2006) that perceived similarity, among partners, in skill level is a significant factor in predicting team compatibility. In all three studies, students rated their compatibility with various partners. They also rated partners on factors such as similarity of skill level, work ethic, programming self-esteem, and time management. Additionally, researchers considered personality type and learning style factors, which they obtained by testing students at the start of the semester. Katira et al. (2004) also found that actual skill level contributed to compatibility in an object-oriented course, but not in CS1 or software engineering, and that partners with different personality types were found to be compatible in CS1 but not in other courses considered. The Williams' study (2006), which seems to subsume the Katira et al.'s (2004) study, did not find that personality type contributed to compatibility, but did find that similar work ethic predicted compatibility in a software engineering course.

Thomas et al. (2003) reported similar results. When students were paired with other students self-reporting equal levels of confidence in programming ability, they recount higher levels of enjoyment and believe they produce better work. Overall, the most self-confident students like pair programming less, especially when paired with students with lower self confidence.

However, Thomas et al. (2003) found discrepancies when students self-reported confidence levels. Students with similar backgrounds and achievement placed themselves at different places on the confidence scale. Grades for students in the most self-confident group were not statistically significantly higher than the class as a whole.

Braught, MacCormick, and Wahls (2010) compared individual programming performance of 259 students in 13 sections of CS1 between 2005 and 2008. Course performance was computed based on individual work, including written homework exercises, weekly programming assignments, three written exams, and two programming exams. The weekly programming assignments were done differently in the 13 different sections analyzed: in 7 sections students were paired by ability, in 2 sections they were paired randomly, and in 4 sections students worked alone. Student ability, for the purpose of pairing, was measured as overall

course performance at the time pairs were created. Individual programming performance was measured by grades earned on the programming exams. Braught et al.'s (2010) results suggest that pairing students by ability has a mild effect on individual programming ability and other individual tasks (written homeworks and tests) for students in the lowest ability quartile.

Watkins and Watkins (2009) used prior course performance to pair students of similar ability. Initially, they used student performance on lab exercises as their measure of ability, but switched to student exam scores later in the course. They found that using lab performance resulted in fewer student-reported problems with their partners.

Radermacher and Walia (2011) paired students based on their major, and found that students in pairs composed of one computing major and one non-computing major perceived pair programming to be less helpful that students in pairs made up of either two computing majors or two non-computing majors. They speculated that this may be due to perceived ability differences between the partners, even though there were no actual differences in the partners' course performance.

Students who were interviewed by Chaparro et al. (2005) expressed the opinion that partners should have similar skill levels. They also suggested that it is not a good idea to pair two novices, because they felt that if neither partner knows what to do they will both struggle and get stuck. This seemingly contradicts the evidence that pair programming is particularly useful for novices, given its success in many introductory programming courses. However, the subjects in this study were post-graduate students, and it is possible that their greater experience influenced their judgment in this area.

The students in Chaparro et al.'s (2005) study also suggested that there should not be a wide disparity in skill level between the two partners. This suggestion was supported by the researchers' observations, in which they noted situations where a skilled student would often take full control of the task with little input from a less skilled, passive partner.

DeClue (2003) reports a similar finding, where students sometimes reported that they felt that working with a less skilled partner slowed them down, or that working with someone more skilled made them feel inferior. Kuppuswami and Vivekanandan (2004) also report that students prefer working with someone with the same or higher level of academic performance.

Although the evidence suggests that most pairs are compatible and that it is important to pair students by ability, there is a paucity of research that examines the impact of minority status on partner compatibility. Katira, Williams, and Osborne (2005) report that minority students are likely to be compatible with their partner, but that pairs in which both students are minorities are more likely to perceive compatibility. Based on their

interviews of 11 African American students, Williams, Layman, Slaten, Berenson, and Seaman (2007) postulate that pair programming appeals to these students due to its collaborative nature. Although these studies suggest that minority status may have an impact on partner compatibility, much more investigation is needed in this area.

3.3. Personality and pairing

Although most pairs have compatible partners, one concern has been the influence of personality on pair compatibility. Some researchers have examined the potential impact of personality traits on pair compatibility and performance.

Choi, Deek, and Im (2008) studied 128 students with no prior programming experience. They created pairs by matching them in different ways based upon each students' Myers-Briggs Type Indicator (MBTI). The MBTI assesses an individual's personality traits based on four scales: sensing/intuitive, thinking/feeling, extraversion/introversion, and judging/perceiving. According to the MBTI model, people have a dominant preference, which will be either their sensing/intuitive or thinking/feeling component. Their auxiliary (or secondary) preference will be the second of these two components. They created pairs whose members were: (1) alike in both their dominant and auxiliary preferences, or (2) opposite in both preferences, or (3) diverse, in that they were alike in one preference but different in the other. As part of the pairing process, students with similar grades were paired in an attempt to control for differing aptitudes.

In their study, pairs worked on two short programming exercises for up to 45 min each. Their solutions were marked on a 10-point scale, which Choi et al. (2008) define as a measure of productivity. There was a significant difference in productivity levels between the three groups; specifically, the pairs made up of students with diverse MBTI traits were more productive than pairs with students who had matched traits. Choi et al. (2008) suggest that this result is due to the diverse pairs' makeup because their common personality trait makes them compatible, but their opposite trait provides a greater perspective to examine more problem solving approaches.

Carver et al. (2007) also used the MBTI to assign partners. In their study, some partners were assigned so that the pairs would have similar personality types, while others were assigned to have different types. They found that the personality types of the pairs had no impact on their study results.

Chao and Atli (2006) identified the most important personality traits for successful pair programming by surveying university students and professional programmers. They found that open-mindedness and

creativity were the most desirable traits for partners, although attentiveness, logical ability, and responsibility were also important.

Fifty-eight students were given a personality trait test and assigned to pairs, based on their open-minded, attentive, logical, and responsible personality traits. Chao and Atli (2006) maximized the variation of the pairs by matching for high/high, high/low, and low/low levels on each personality trait. The pairs were given a programming task, which was evaluated for the quality of the resulting code, specifically correctness of output, documentation, style, correct object usage, and the user interface. They found no evidence that matching pairs on personality traits made a difference in the quality of the code produced. They also found no correlation between personality traits and compatibility or enjoyment as reported by the participants.

Salleh, Mendes, Grundy, and Burch (2009) also investigated the role of personality in successful pair programming. In their study, students' personality characteristics were determined using the five-factor model, which is based on five broad personality traits: openness to experience, conscientiousness, extraversion, agreeableness, and neuroticism. They formed pairs with both similar and mixed levels of conscientiousness. The pairs completed short (about 75 min) laboratory exercises.

There was no evidence that personality traits affected student academic performance. There were no significant differences in assignment or test scores between students who were in mixed or same personality pairs. Based on survey results, they also found that personality had no impact on student satisfaction or confidence. Students in both types of pairs were highly confident (87.9% of students reported that they were highly confident in their solutions to the programming assignments), and 92.6% of them enjoyed working with their partner.

Taken together, these studies suggest that the personality traits of the individual partners in a pair may impact that pair's compatibility and performance, but it appears that this effect may be small. However, as noted by Salleh, Mendes, and Grundy (2011), these results may be partially attributed to the variety of instruments used to measure personality.

3.4. Transition to solo programming

Although there is substantial empirical evidence to the contrary (as discussed in Section 2.2), some faculty members have expressed concern that students who pair program in their first course would not be able to program individually in later courses. Jacobson and Schaefer (2008) had to overcome this objection from colleagues at their institution, but found that "pair programming did not interfere with a student's capability to program individually" in later courses. They recommend that students in

the later courses be given clear instructions about appropriate and inappropriate collaboration.

Students at two institutions who were interviewed about the transition from paired to solo programming (Simon & Hanks, 2008) indicated that pair programming helped them learn to program and gave them a good foundation for when they had to program without a partner. As stated by one of the students, ''Programming in pairs helped me a lot, so when it came time to program by myself, I was ready'' (Simon & Hanks, 2008, p. 23).

4. Practical matters

Research suggests that pair programming is a successful technique that provides pedagogical and social benefits for students. But, how does one go about using pair programming so that students can enjoy these benefits without overloading the teaching staff? Specifically, how do you teach the students how to pair? How do you assign partners so that the pairs will be successful? And, how do you assess students individually if they are working collaboratively? Three papers that specifically address these questions are Bevan et al. (2002), Jacobson and Schaefer (2008), and Williams, McCrickard, Layman, & Hussein (2008).

Based on 7 years of experience with more than 1000 students, Williams et al. (2008) outline guidelines for implementing pair programming in the classroom. They contend that:

- Students need training in order to successfully pair. In particular, freshman and sophomores should not be paired unless there is supervised lab time available.
- Teachers should check in with pairs to make sure they are working effectively together and to address questions as they arise during the lab exercises. However, pairs should be encouraged to find answers independently as much as possible.
- Attendance should be required and penalties for tardiness should be applied to protect one partner from another who is less diligent.
- Similarly, instructors should ask students for feedback about their partners and take prompt action when one partner is not performing equitably. This is particularly important when students are paired outside of a closed laboratory.
- By the same token, students must report problems with their partners immediately so the instructor can intervene.
- Students should be evaluated on both paired and individual work.
- Instructors should make every attempt to assign compatible partners. Williams et al. (2008) noted that students prefer to work with a partner of equal or better skill level and dysfunctional pairs occurred when there were very different work ethics and/or skill

levels. In addition, students should be assigned to different partners throughout the term.
- Environmental factors are important. Pairs should be able to sit comfortably next to each other as they work with both partners having easy access to the screen, keyboard, and mouse.
- Finally, the pair should be working together on the same problem.

Williams et al. (2008) created a peer evaluation system, PairEval[1], which instructors can use to observe rating trends for each student. Their pair programming training video[2] is also available for students and instructors.

Bevan et al. (2002) additionally suggested instituting a coding standard as novice programmers tend to believe that their style is the ''right'' style. By enforcing an instructor-defined coding standard, arguments about trivial issues are minimized.

4.1. Partner assignment and rotation

There are various mechanisms for assigning partners, many addressing issues related to compatibility. There is also the issue of whether partner assignments should be maintained for the duration of a course or changed regularly, and if so, how frequently.

Many instructors allow students to self-select their partners. This may be the simplest technique for large courses, or for situations where pairing is used only in closed labs. In the pair programming study conducted by Chaparro et al. (2005), students paired in closed labs only and selected their own partners for each of the three paired labs.

McDowell et al. (2003) asked students to list up to three potential partners with whom they would like to be paired. Nearly every student was assigned to a partner on his or her list. Students who did not list any potential partners were given one randomly. Students worked with the same partner for the duration of the course (10 weeks), although a few students had to be given new partners due to course drops or schedule changes.

As discussed in Section 3.2, most pairs are compatible, but it is better if the partners have similar skill levels. To help facilitate these types of pairings, some instructors wait for a few weeks to allow students to get to know each other better, so that they can choose a compatible partner (Jacobson & Schaefer, 2008). As Jacobson and Schaefer (2008) note, ''students tend to seek out partners who they perceive have a skill level at least as high as their own, and with whom they believe they can work'' (p. 95).

In other institutions, students switch partners at various times during the term. Williams et al. (2008) discuss pair rotation as one of their guidelines, as it allows students to meet more of their peers, and it

prevents students from getting stuck with an underperformer or non-contributor for an entire term. They also argue that it is beneficial for the teaching staff because a more diverse set of peer evaluations provides a more accurate assessment of each student's work.

Srikanth, Williams, Wiebe, Miller, and Balik (2004) surveyed students by asking, "Do you think it was a good idea to change partners after each assignment?" Of 270 CS1 students, 197 (73%) agreed that pair rotation was a good idea; 16 out of 17 (94%) software engineering students also agreed. Students reported that pair rotation gave exposure to more classmates and allowed them to have a new partner when there were partner incompatibilities. But when they had a highly compatible partner, pair rotation led to the potential for a less compatible partner. They also found inefficiencies as they had to adapt to a new partner.

Student pairs in DeClue's (2003) study worked on a 6-week project broken into three 2-week phases. Students traded partners after each phase, with one partner continuing on the project and one transferring to a different project, forcing them to work on unfamiliar code. Declue's students reported that this trading process made them appreciate the value of good code design and documentation, which is something that many students tend to disregard otherwise.

Although Carver et al. (2007) initially assigned partners based on their Myers-Briggs personality type, they only did this because they did not have any information about their students' technical competency. When they reassigned partners later in the course, they did it based on student performance to that point, because they felt that the literature showed that this would produce the most compatible pairs.

VanDeGrift (2004) also altered the criteria for assigning pairs as the term progressed: using random pairings for the first assignment, comparable skill level on the second, and comparable effort level on the third, although how these levels are assessed is not described in the paper.

It seems clear that switching partners provides some benefits for students as partners can be assigned based on skill, students avoid being stuck with an incompatible or mismatched partner, and students get exposed to more classmates and differing working styles. These benefits must be weighed against downsides such as increased work for the instructor and issues related to scheduling – just as students work out a schedule with one partner, they have to switch to work with another. Of course, this is not an issue in situations where pair programming is only used in closed labs.

4.2. Teaching students to pair

Many different approaches have been taken to teach students how to pair, including assigned readings, lectures, and practice lab sessions. Bevan et al. (2002) recommend introductory activities such as the ones described

in this section, and also suggests candidly addressing scheduling compatibility and student concerns about working with a partner.

Students in the pairing sections of McDowell et al. (2003) were given a 15–20 min description of pair programming. They were also asked to read Willams and Kessler's ''Kindergarten'' paper (2000). To encourage them to read the paper, students were told that the first quiz might include a question about it. Students in VanDeGrift's (2004) study also read the ''Kindergarten'' paper, and the course instructors gave a pair programming demonstration, followed by a debriefing of the navigator and driver roles.

To prepare students for pair programming, Declue (2003) spent a portion of one lecture describing it and discussing its purpose. Additionally, ''one 100 minute lab period was devoted to practicing pair programming'' (DeClue, 2003, p. 50), and further guidance was provided with laboratory instructions.

Kuppuswami and Vivekanandan (2004) gave students in the paired groups a 1-hour lecture about pair programming, in which they were taught about the roles. They also practiced pair programming in one laboratory session before the experiment was conducted. Chaparro et al. (2005) gave students a brief explanation of pair programming, which focused on the characteristics of good collaboration and the driver/navigator roles.

Although the amount of training conducted in these studies tended to be quite small, pair programming had a positive impact on various aspects of student performance and learning. On the other hand, paired students in the study conducted by Somervell (2006) did not benefit from pairing. These students were not provided with any background or training. Instead, ''Students were left alone in defining roles and how each student would perform in these roles'' (Somervell, 2006, p. 307). As Somervell acknowledges, perhaps students need to receive some instruction in how to pair in order for it to be beneficial to them. This observation is echoed by Williams and Kessler (2001, p. 19), who note that, ''It is very important to provide the students some class/lab time to work with their partner,'' as this allows the partners to bond, and gives instructors and lab assistants an opportunity to observe and train the students as they pair.

Williams et al. (2008) note in their guidelines for successful pairing that it is necessary to explicitly train students to pair and that this is best done in a supervised lab setting. They provide a 15-min training video to help students understand the driver and navigator roles, the need to switch roles and the need to be actively participating at all times. The lab instructor closely observes the pairs, intervening as necessary to make sure the partners share the work and the roles appropriately. They state, ''We strongly advise against pairing first or second year undergraduate students if no pair programming will occur in a closed lab or classroom setting that is monitored by a member of the teaching staff'' (Williams et al., 2008, p. 447). They observe that starting out as pairs in a structured

environment will encourage students to feel more comfortable when meeting together outside of class.

Even with training, it is sometimes necessary to remind students to switch roles (Jacobson & Schaefer, 2008). In closed labs, this can be done by the instructor or teaching assistant at regular intervals. This helps students to learn how to pair, and gives both partners an opportunity to drive. It may also help avoid other pairing problems, such as those discovered by Bevan et al. (2002). From written student logs and teaching assistant observations, Bevan et al. discovered that some pairs used a ''divide and conquer'' approach, while others alternated development by emailing the latest version back and forth. Some partners did not want to drive at all. Moreover, a willingness to submit work with both partner's names, even if one partner had not contributed, was uncovered.

Although it is important to teach students how to pair, instructors should not be overly concerned when students do not rigorously follow the driver and navigator roles discussed in much of the literature. Early descriptions of pair programming emphasized the differences between the two roles, and argued that the navigator operated at a higher level of abstraction than the driver. Recent observational studies have shown that the roles are much less distinct. For example, Bryant, Romero, and du Boulay (2008) observed pairs of professional programmers and found that there were no significant differences between the two partners in terms of the types of statements they made or the level of abstraction they engaged in. They suggest that pair programming may in fact be more of a ''tag team'' approach to software development, in which the two partners share the cognitive load of software development.

Chaparro et al. (2005) made similar observations when studying student pair programmers. They noted that it was often difficult for them to determine which student was playing which role, and observed situations in which one student was controlling the keyboard while the other was controlling the mouse. They also observed other types of role behaviors, such as ''teacher-learner'' or ''thinker-doer.''

Chong and Hurlbutt (2007) also found no distinct division of labor between the two partners, and referred to the ''driver/navigator myth'' (p. 357). In the pairs they observed, both partners contributed to discussions equally and frequently switched control of the keyboard and mouse. They suggest that emphasizing the driver and navigator roles when teaching pair programming ''may actually run counter to the ways that pairs work most naturally and effectively'' (Chong & Hurlbutt, 2007, p. 354).

4.3. Assessing individual contributions

Clearly, educators want their students to know the course material when the course is over. Some educators may be concerned that students who

pair program may not gain the essential knowledge, or that it will be difficult to assess individual learning in a paired environment. Many different techniques have been used to assess individual student's learning. These include individual lab exams, survey instruments, quiz questions about the students' shared work, written reports, and interviews.

Preston (2006) makes some recommendations (based on collaborative learning research) for implementing pair programming. He argues that individual accountability is critical to ensure that every student learns the material that is intended to be taught through a collaborative activity. Preston (2006) makes two recommendations to encourage individual accountability with pair programming (p. 19):

(1) Tests should require students to develop code, to interpret code, or both.
(2) Test scores for individuals should be more heavily weighted than joint project scores when determining the final grade.

Preston argues that these two recommendations will encourage students to stay actively involved while working with their partners, and to become more concerned with individual learning because they know they will be assessed on their understanding.

For example, the students in the CS1 course described by Jacobson and Schaefer (2008) worked in pairs on lab exercises, but took individual lab exams to assess their learning. Similarly, students in the CS1 courses studied by McDowell et al. (2006) were assessed individually using exams, which specifically tested the students' ability to write code.

Williams et al. (2008) note in their guidelines for successful pairing that students should be assessed using both individual and paired work, as this helps prevent over-reliance on one partner. In their view, the percentage of the course grade based on collaborative work should vary from course to course, based on the type of course. For example, in their CS1 course, only 10% of the grade is based on paired work, while in a software engineering course, a much greater proportion of the grade is allocated to group work.

Students can also assess their partner's work. For each of their programming assignments, students at one of the institutions studied by Simon and Hanks (2008) filled out forms indicating the amount of time they spent in the driver and navigator roles as well as the amount of time they spent working alone. They also assessed the contribution made to the program by their partner. DeClue (2003) asked students to indicate the percentage of work that was contributed by each partner at the end of every 2-week phase in their projects. He potentially adjusted students' individual grades based on their responses to this question. DeClue (2003) also included questions on quizzes that were directly

related to the projects, such as "What is the name of the class which contains your hash method?".

Some instructors use pair programming in closed labs only, where the pairs can be easily supervised (e.g. Carver et al., 2007; Chaparro et al., 2005). All other work, including larger programming assignments, is done by students working alone. This approach attempts to ensure that students are able to program, and do not simply get a "free ride" from their partners. However, as found by Simon and Hanks (2008), it is somewhat naïve to assume that students are not getting help from others, even when they have specifically been told to work alone.

VanDeGrift (2004) combined pair programming projects with individual written reports. The reports served as a mechanism for accountability in a CS1 class with more than 500 students which did not include a closed laboratory component. The reports, which were weighted equally with the three projects, required students to reflect individually on the project purpose, user interface, design process, system internals, test cases, and what they learned. A survey given at the end of the term revealed that while students valued the chance to explain their projects, they felt the reports were too time consuming and seemed like "busy" work. But, the written reports appeared to reduce the concern that lazy partners were earning "free" grades as fewer than 5% of students reported this as a problem.

Cliburn (2003) used peer evaluations in which students emailed the instructor the percentage of effort contributed by each partner to the project (e.g. "John 50%, Mary 50%"). Lower grades were assigned to students whose partners reported that they did not contribute equally to the project. This seemed to eliminate complaints about students who were regarded as "parasites" for relying on their partners to do all the work.

Students in the courses described by Williams et al. (2008) used the PairEval tool to rate their partner after they turned in each programming assignment. PairEval allows students to choose a word, such as "Excellent," "Satisfactory," or "Deficient," to describe their partner's contribution on an assignment. Based on the rating, a student's grade can be adjusted to reflect their level of participation.

Another technique that can be used to alleviate concerns that one partner might not be contributing his or her share of the work is to have the students individually explain their code. For example, 10% of the grade on programming assignments for students at one of the institutions studied by Simon and Hanks (2008) was based on their ability to describe their work in individual interviews.

4.4. Physical setup

Pair programming is typically done by having the partners share a single workstation. However, many variations of this configuration have been used with success.

Students in Müller and Tichy's (2001) extreme programming course tended to use two adjacent computers while pair programming. They used one of the displays for program development and the second for accessing Java documentation on the internet. Three-quarters of the students felt that having two displays was better than having only one.

Few authors have described their physical lab settings in detail. However, Williams et al. (2008) describes several alternatives. In their view, a traditional lab environment is acceptable if there is enough room to allow students to switch roles without switching chairs. A less satisfactory arrangement, when no suitable lab environment is available, is to have pairs of students with laptop computers move desks to allow them to sit next to each other. Williams et al. (2008) also describe the ideal lab environment as one in which every computer has two monitors, mice, and keyboards, thus making it very easy for students to pair.

One potential downside of using pair programming is the collocation requirement, as both partners must find a common time and place to meet. This can be challenging for students who live far from campus. Extending the pair programming model to allow the partners to virtually meet from separate locations mitigates this issue, making it possible for more students to pair.

An early study of distributed pair programming was conducted by Baheti, Gehringer, and Stotts (2002). They studied a graduate-level object-oriented programming course in which students worked on team projects. Students in 21 teams used pair programming; 5 of these teams used distributed pairing, in which the partners used a screen-sharing tool to allow them to virtually pair from separate locations. They found no differences between the collocated and distributed teams in terms of productivity or software quality. They concluded that, "Distributed pair programming in virtual teams is a feasible way of developing object-oriented software" (Baheti et al., 2002, p. 219).

Hanks (2008a) developed and empirically evaluated a tool specifically designed to support distributed pairing. This tool provided a second cursor to allow both partners to gesture effectively. In his study, paired students in three sections of a CS1 course were randomly assigned to two groups. The students in the first group of pairs had to physically meet to work on their programming assignments, while the students in the second group were allowed to use the distributed pairing tool to virtually pair from separate locations.

Students in both groups performed equivalently in the class, in terms of grades on programming assignments and the final exam. There was some evidence that the students who were allowed to pair from separate locations spent a smaller proportion of their time working alone, thus potentially accruing more of the benefits associated with pair

programming. Together, the Baheti et al.'s (2002) and Hanks, (2008a) studies indicate that distributed pairing can work in educational settings.

5. Concluding remarks

Pair programming has been used in the classroom for nearly 10 years, and its application in educational settings thus far indicates pair programming offers a range of benefits for students with minimal challenges. However, the evidence of actual improvements in learning by students (as measured objectively and individually) is scarce. As Salleh et al. (2011) note in their systematic literature review, there is evidence that pair programming is effective in the classroom, but much more research is needed. There are no widespread studies, multi-institutional studies or longitudinal studies, and very few replicated studies. Little is also understood about how best to teach students to pair program.

The majority of the research into the educational application of pair programming has focused on measurable benefits such as course success, retention, program quality, confidence, and enjoyment. Appropriately, that work has assessed pair programming using quantitative research methods evaluating various-sized groups of subjects assigned to "control" and "experimental" groups. These studies have shown pair programming to be generally beneficial for university level students, particularly in introductory courses, although results attributable to pair programming have ranged from no impact to large statistically significant improvements. Further research in these areas is needed to better characterize the situations in which pair programming leads to improved student performance.

In considering the benefits of pair programming, several important groups have largely been neglected or only considered as an afterthought. These groups include women, ethnic minorities, and pre-college students. Some questions to consider are:

- Is pair programming beneficial for women or minority students? If so, in what ways and why?
- What impacts might pair programming have in K-12 education? Is it workable beyond the classroom lab in non-adult settings?

Another area of research includes studies that focus on how to effectively implement pair programming. Many have found that collaboration strategies other than the strict driver-navigator scenario work best in some settings. Also, looking beyond collaboration strategies, researchers have studied how best to assign partners, how to assess individual knowledge and contributions, and how best to teach students to pair. Some questions to consider are:

- Are there particular benefits to strictly defined roles? If so, is there a best pair-role scenario? Or might less strictly defined pair strategies be equally as effective? What impact does frequency of role switching play?
- It appears that pairing students by actual or perceived ability level leads to improved partner compatibility. Is this true in general? What factors influence this? Should students be paired by ability level?
- Personality seems to have little impact on pair compatibility, but as Salleh et al. (2011) argue, these results may be attributable to the instruments used to measure personality. They suggest that more research is needed in this area.
- Can pair programming in computer labs be informed by pair work in chemistry and physics labs?

Attitude plays an important role in so many aspects of life; one expects it plays an important role in learning as well. In regards to pair programming:

- What role does attitude play? Can a statistically significant correlation between attitude and performance be found?
- Why do some students strongly resist collaboration? Do these students share certain characteristics? Is it acceptable or even better for these students to work alone? And if so, how can pairs and individuals working alone be managed?
- Can bad performance results for the Warrior/Phobe or major/non-major pairing be replicated and/or explained?

Only one study considered pair programming in a distance education setting. In light of the continued growth and ever-evolving need for distance education, further study of pair programming in these settings is needed. Tools and techniques to aid pair programming for distance education need to be expanded and evaluated.

The research that seems most lacking includes studies designed to shed light on *how* and *why* pair programming works or does not work. One obvious reason is that these are typically smaller, qualitative studies, and CS educators are naturally more comfortable with quantitative research. To understand the hows and whys of pair programming will require educators to borrow from and draw relationships to other disciplines such as educational psychology, mathematics and science education, and research on expertise. This type of research will allow for richer questions such as what types of discourse do pairs engage in, how is the cognitive load shared between partners, or how does the act of explaining one's reasoning to a peer facilitate learning. These issues appear to be relevant

to the effective use of pair programming in the classroom. Some of these issues are investigated in Murphy, Fitzgerald, Hanks, and McCauley (2010), but these results are only preliminary and should lead to further investigation.

As in all areas of teaching and learning, research results provide data which fuels innovation. The research into pair programming is encouraging; however, replication is necessary to verify findings. Research into additional questions, such as those listed in this section, is encouraged.

Notes

1. See http://agile.csc.ncsu.edu/pairlearning/paireval.php for more information.
2. Available at http://agile.csc.ncsu.edu/pairlearning/educators.php#ppvideo.

References

Ackerman, A.F., Buchwald, L.S., & Lewski, F.H. (1989). Software inspections: An effective verification process. *IEEE Software*, *6*(3), 31–36.

Baheti, P., Gehringer, E., & Stotts, D. (2002). Exploring the efficacy of distributed pair programming. In *Extreme programming and Agile methods - XP/Agile Universe 2002*, no. 2418 in LNCS (pp. 208–220). Berlin/Heidelberg: Springer.

Bandura, A. (1977). Self-efficacy: Toward a unifying theory of behavioral change. *Psychological Review*, *84*, 191–215.

Beck, K. (2000). *Extreme programming explained: Embrace change*. Reading, MA: Addison-Wesley.

Begel, A., & Nagappan, N. (2008). Pair programming: What's in it for me? In *ESEM '08: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement* (pp. 120–128). Kaiserslautern, Germany: ACM.

Bennedsen, J., & Caspersen, M.E. (2007). Failure rates in introductory programming. *SIGCSE Bulletin*, *39*, 32–36.

Bevan, J., Werner, L., & McDowell, C. (2002). Guidelines for the use of pair programming in a freshman programming class. In *Proceedings of the 15th conference on software engineering education and training, February 25–27* (pp. 100–108). Washington, DC: IEEE Computer Society.

Bipp, T., Lepper, A., & Schmedding, D. (2008). Pair programming in software development teams – An empirical study of its benefits. *Information and Software Technology*, *50*, 231–240.

Braught, G., Eby, L.M., & Wahls, T. (2008). The effects of pair-programming on individual programming skill. In *SIGCSE '08: Proceedings of the 39th SIGCSE technical symposium on computer science education* (pp. 200–204). New York, NY: ACM.

Braught, G., MacCormick, J., & Wahls, T. (2010). The benefits of pairing by ability. In *SIGCSE '10: Proceedings of the 41st ACM technical symposium on computer science education* (pp. 249–253). Milwaukee, WI: ACM.

Braught, G., Wahls, T., & Eby, M. (2011). The case for pair programming in the computer science classroom. *ACM Transactions on Computing Education*, *11*(1), 2:1–2:21.

Bryant, S., Romero, P., & du Boulay, B. (2008). Pair programming and the mysterious role of the navigator. *International Journal of Human–Computer Studies*, *66*, 519–529.

Cantwell Wilson, B. (2002). A study of factors promoting success in computer science including gender differences. *Computer Science Education*, *12*(1/2), 141.

Cao, L., & Xu, P. (2005). Activity patterns of pair programming. In *HICSS '05: Proceedings of the 38th annual Hawaii international conference on system sciences (HICSS'05) - Track 3* (pp. 88a). Los Alamitos, CA: IEEE Computer Society.

Carver, J.C., Henderson, L., He, L., Hodges, J., & Reese, D. (2007). Increased retention of early computer science and software engineering students using pair programming. In *CSEET '07: Proceedings of the 20th conference on software engineering education & training* (pp. 115–122). Washington, DC: IEEE Computer Society.

Chao, J., & Atli, G. (2006). Critical personality traits in successful pair programming. In *AGILE '06: Proceedings of the conference on AGILE 2006* (pp. 89–93). Washington, DC: IEEE Computer Society.

Chaparro, E.A., Yuksel, A., Romero, P., & Bryant, S. (2005). Factors affecting the perceived effectiveness of pair programming in higher education. In *Proceedings of the 17th workshop of the Psychology of Programming Interest Group* (pp. 5–18). Retrieved from http://www.ppig.org/workshops/17th-programme.html

Choi, K.S., Deek, F.P., & Im, I. (2008). Exploring the underlying aspects of pair programming: The impact of personality. *Information and Software Technology*, *50*, 1114–1126.

Chong, J., & Hurlbutt, T. (2007). The social dynamics of pair programming. In *ICSE '07: Proceedings of the 29th international conference on software engineering* (pp. 354–363). Washington, DC: IEEE Computer Society.

Cliburn, D.C. (2003). Experiences with pair programming at a small college. *Journal of Computing in Small Colleges*, *19*(1), 20–29.

DeClue, T.H. (2003). Pair programming and pair trading: Effects on learning and motivation in a CS2 course. *Journal of Computing in Small Colleges*, *18*, 49–56.

Hanks, B. (2005). *Empirical studies of distributed pair programming*. Doctoral dissertation, University of California, Santa Cruz.

Hanks, B. (2006). Student attitudes toward pair programming. In *Proceedings of the 11th annual conference on innovation and technology in computer science education (ITiCSE 2006), June 26–28* (pp. 113–117). New York, NY: ACM.

Hanks, B. (2008a). Empirical evaluation of distributed pair programming. *International Journal of Human–Computer Studies*, *66*, 530–544.

Hanks, B. (2008b). Problems encountered by novice pair programmers. *Journal on Educational Resources in Computing*, *7*, 1–13.

Hanks, B., McDowell, C., Draper, D., & Krnjajic, M. (2004). Program quality with pair programming in CS1. In *ITiCSE '04: Proceedings of the 9th annual SIGCSE conference on innovation and technology in computer science education* (pp. 176–180). New York, NY: ACM.

Jacobson, N., & Schaefer, S.K. (2008). Pair programming in CS1: Overcoming objections to its adoption. *SIGCSE Bulletin*, *40*, 93–96.

Jensen, R.W. (2003). A pair programming experience. *CrossTalk, The Journal of Defense Software Engineering*. Retrieved from www.crosstalkonline.org/storage/issue-archives/2003/200303/200303-Jensen.pdf

Katira, N., Williams, L., Wiebe, E., Miller, C., Balik, S., & Gehringer, E. (2004). On understanding compatibility of student pair programmers. In *Proceedings of the thirty-fifth SIGCSE technical symposium on computer science education* (pp. 7–11). New York, NY: ACM.

Katira, N., Williams, L., & Osborne, J. (2005). Towards increasing the compatibility of student pair programmers. In *Proceedings of the 27th international conference on software engineering* (pp. 625–626). Washington, DC: IEEE Computer Society.

Kuppuswami, S., & Vivekanandan, K. (2004). The effects of pair programming on learning efficiency in short programming assignments. *Informatics in Education*, *3*, 251–266.

Margolis, J., & Fisher, A. (2002). *Unlocking the clubhouse: Women in computing*. Cambridge, MA: MIT Press.

McCabe, T. (1976). A complexity measure. *IEEE Transactions on Software Engineering, SE-2*, 308–320.

McDowell, C., Werner, L., Bullock, H., & Fernald, J. (2003). The impact of pair programming on student performance, perception and persistence. In *Proceedings of the international conference on software engineering (ICSE 2003), May 3–10* (pp. 602–607). Washington, DC: IEEE Computer Society.

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

Melnik, G., & Maurer, F. (2002). Perceptions of agile practices: A student survey. In *Extreme programming and agile methods – XP/Agile Universe 2002* (pp. 241–250). No. 2418 in LNCS. Berlin/Heidelberg: Springer.

Mendes, E., Al-Fakhri, L., & Luxton-Reilly, A. (2006). A replicated experiment of pair-programming in a 2nd-year software development and design computer science course. In *Proceedings of the 11th annual conference on innovation and technology in computer science education (ITiCSE 2006), June 26–28* (pp. 113–117). New York, NY: ACM.

Mendes, E., Al-Fakhri, L.B., & Luxton-Reilly, A. (2005). Investigating pair-programming in a 2nd-year software development and design computer science course. In *ITiCSE '05: Proceedings of the 10th annual SIGCSE conference on innovation and technology in computer science education* (pp. 296–300). New York, NY: ACM.

Müller, M.M., & Tichy, W.F. (2001). Case study: Extreme programming in a university environment. In *Proceedings of the 23rd international conference on software engineering* (pp. 537–544). New York, NY: ACM.

Murphy, L., Fitzgerald, S., Hanks, B., & McCauley, R. (2010). Pair debugging: A transactive discourse analysis. In *ICER '10: Proceedings of the sixth international workshop on computing education research* (pp. 51–58). Denmark: Aarhus.

Nagappan, N., Williams, L., Wiebe, E., Miller, C., Balik, S., Ferzli, M., et al. (2003). Pair learning: With an eye toward future success. In *Extreme programming and agile methods – XP/Agile Universe 2003* (pp. 185–198). No. 2753 in LNCS. Berlin/Heidelberg: Springer.

Nosek, J.T. (1998). The case for collaborative programming. *Communications of the ACM*, *41*, 105–108.

Phongpaibul, M., & Boehm, B. (2006). An empirical comparison between pair development and software inspection in Thailand. In *ISESE '06: Proceedings of the 2006 ACM/IEEE international symposium on empirical software engineering* (pp. 85–94). New York, NY: ACM.

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

Radermacher, A., & Walia, G. (2011). Investigating the effective implementation of pair programming: An empirical investigation. In *SIGCSE '11: Proceedings of the 42nd ACM technical symposium on computer science education* (pp. 655–660). Dallas, Texas: ACM.

Ramalingam, V., LaBelle, D., & Wiedenbeck, S. (2004). Self-efficacy and mental models in learning to program. In *ITiCSE '04: Proceedings of the 9th annual SIGCSE conference on innovation and technology in computer science education* (pp. 171–175). New York, NY: ACM.

Salleh, N., Mendes, E., Grundy, J., & Burch, G.S.J. (2009). An empirical study of the effects of personality in pair programming using the five-factor model. In *ESEM '09: Proceedings of the 2009 3rd international symposium on empirical software engineering and measurement* (pp. 214–225). Washington, DC: IEEE Computer Society.

Salleh, N., Mendes, E., & Grundy, J. (2011). Empirical studies of pair programming for CS/SE teaching in higher education: A systematic literature review. *IEEE Transactions on Software Engineering*, *PP*(99), 1.

Sennett, J., & Sherriff, M. (2010). Compatibility of partnered students in computer science education. In *SIGCSE '10: Proceedings of the 41st ACM technical symposium on computer science education* (pp. 244–248). Milwaukee, WI: ACM.

Simon, B., & Hanks, B. (2008). First-year students' impressions of pair programming in CS1. *Journal of Educational Resources in Computing*, *7*, 1–28.

Somervell, J. (2006). Pair programming: Not for everyone? In *Proceedings of the 2006 international conference on frontiers in education: Computer science and computer engineering (FECS'06)* (pp. 303–307). Las Vegas, NV: CSREA Press.

Srikanth, H., Williams, L., Wiebe, E., Miller, C., & Balik, S. (2004). On pair rotation in the computer science course. In *CSEET '04: Proceedings of the 17th conference on software engineering education and training* (pp. 144–149). Washington, DC: IEEE Computer Society.

Thomas, L., Ratcliffe, M., & Robertson, A. (2003). Code warriors and Code-A-Phobes: A study in attitude and pair programming. In *Proceedings of the 34th SIGCSE technical symposium on computer science education* (pp. 363–367). New York, NY: ACM.

VanDeGrift, T. (2004). Coupling pair programming and writing: Learning about students' perceptions and processes. In *Proceedings of the thirty-fifth SIGCSE technical symposium on computer science education* (pp. 2–6). New York, NY: ACM.

Watkins, K.Z.B., & Watkins, M.J. (2009). Towards minimizing pair incompatibilities to help retain under-represented groups in beginning programming courses using pair programming. *Journal on Computing in Small Colleges*, *25*, 221–227.

Werner, L., Hanks, B., & McDowell, C. (2004). Pair programming helps female computer science students. *ACM Journal of Educational Resources in Computing*, *4*(1).

Wiedenbeck, S. (2005). Factors affecting the success of non-majors in learning to program. In *ICER '05: Proceedings of the first international workshop on computing education research* (pp. 13–24). New York, NY: ACM.

Williams, L., & Kessler, R. (2000). All I really need to know about pair programming I learned in kindergarten. *Communications of the ACM*, *43*, 108–114.

Williams, L., & Kessler, R. (2001). Experiments with industry's "pair-programming" model in the computer science classroom. *Computer Science Education*, *11*(1), 7–20.

Williams, L., Layman, L., Osborne, J., & Katira, N. (2006). Examining the compatibility of student pair programmers. In *AGILE '06: Proceedings of the conference on AGILE 2006* (pp. 411–420). Washington, DC: IEEE Computer Society.

Williams, L., Layman, L., Slaten, K., Berenson, S., & Seaman, C. (2007). On the impact of a collaborative pedagogy on African American millennial students in software engineering. In *Proceedings of the 29th international conference on software engineering* (pp. 677–687). Washington, DC: IEEE Computer Society.

Williams, L., McCrickard, S.D., Layman, L., & Hussein, K. (2008). Eleven guidelines for implementing pair programming in the classroom. In *Proceedings of the Agile 2008 conference* (pp. 445–452). Washington, DC: IEEE Computer Society.

Williams, L.A. (2000). *The collaborative software process*. Doctoral dissertation, University of Utah, Salt Lake City, UT.

Appendix

	Success in first- and second-year programming courses	Success in Later courses	Retention	Quality of programs produced by pairs	Confidence in solutions	Attitudes toward pairing	Impact on learning	Social aspects of pairing	Impact on women	Scheduling	Partner compatibility	Personality and Pairing	Transition to solo programming	Partners assignment and rotation	Teaching Students to pair	Assessing individual contributions	Physical setup
Braught, Eby, and Wahls (2008)	X															X	
McDowell, Werner, Bullock, and Fernald (2003)	X			X	X	X								X	X		
McDowell, Werner, Bullock, and Fernald (2006)	X	X	X													X	
Mendes, Al-Fakhri, and Luxton-Reilly (2005)	X					X											
Nagappan, Willams, Wiebe, Miller, Balik, Ferzli, and Petlick (2003)	X	X					X										
Somervell (2006)	X							X							X		
Jacobson and Schaefer (2008)		X											X		X	X	
Carver, Henderson, He, Hodges, and Reese (2007)			X				X		X		X	X		X		X	
Begel and Nagappan (2008)				X						X							
Bipp, Lepper, and Schmedding (2008)				X			X										
DeClue (2003)				X	X		X			X	X			X	X	X	
Hanks, McDowell, Draper, and Krnjajic (2004)				X	X												
Melnik and Maurer (2002)				X						X							
Phongpaibul and Boehm (2006)				X													
Williams and Kessler (2001)				X											X		
Braught, MacCormick, and Wahls (2010)					X	X					X						
Braught, Wahls, and Eby (2011)					X	X			X								
Müller and Tichy (2001)					X		X										X
Nosek (1998)					X	X											
Simon and Hanks (2008)					X		X	X		X			X			X	
VanDeGrift (2004)					X	X	X		X	X				X	X	X	
Williams (2000)					X	X											
Chaparro, Yuksel, Romero, and Bryant (2005)						X	X				X			X	X	X	
Hanks (2006)						X	X		X								
Thomas, Ratcliffe, and Robertson (2003)						X	X				X						
Cao and Xu (2005)							X			X	X						
Hanks (2008)							X										
Kuppuswami and Vivekanandan (2004)							X				X				X		
Cliburn (2003)								X								X	
Jacobson and Schaefer (2008)								X			X		X	X			
Werner, Hanks, and McDowell (2004)									X								
Bevan, Werner, and McDowell (2002)											X				X		
Hanks (2005)											X						

(*continued*)

Appendix. (*Continued*).

	Success in first- and second-year programming courses	Success in Later courses	Retention	Quality of programs produced by pairs	Confidence in solutions	Attitudes toward pairing	Impact on learning	Social aspects of pairing	Impact on women	Scheduling	Partner compatibility	Personality and Pairing	Transition to solo programming	Partners assignment and rotation	Teaching Students to pair	Assessing individual contributions	Physical setup
Katira, Williams, Wiebe, Miller, Balik, and Gehringer (2004)											X						
Radermacher and Walia (2011)											X						
Sennett and Sherriff (2010)											X						
Williams, Layman, Osborne, and Katira (2006)											X						
Chao and Atli (2006)												X					
Choi, Deek, and Im (2008)												X					
Williams, McCrickard, Layman, and Hussein (2008)														X	X	X	X
Srikanth, Williams, Wiebe, Miller, and Balik (2004)														X			
Williams and Kessler (2000)															X		
Preston (2006)																X	