Pair Programming in Perspective: Its Impact on Academic Achievement, Persistence, and Equity in Computer Science

Lindsay Jarratt, Nicholas A. Bowman, KC Culver, and Alberto M. Segre

University of Iowa

Paper prepared for the 2019 annual meeting of the Association for Education Finance and Policy, Kansas City, MO

Correspondence concerning this article should be addressed to Lindsay Jarratt, Department of Educational Policy and Leadership Studies, University of Iowa, N440 Lindquist Center, Iowa City, IA 52242, lindsay-jarratt@uiowa.edu. This material is based upon work supported by the National Science Foundation under Grant No. 1611908. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Pair Programming in Perspective: Its Impact on Academic Achievement, Persistence, and Equity in Computer Science

Abstract

Pair programming is a form of collaborative learning in computer science that involves two students (or employees) working together collectively on the same coding project. Previous research has largely identified positive outcomes associated with pair programming that range from course grades to the quality of the resulting code. Pair programming may also facilitate interactions that improve the climate for women and Students of Color, thereby reducing equity gaps in academic achievement and persistence in computer science. However, the existing research findings are not consistently positive, perhaps because pair programming may often fall short of facilitating an ideal learning environment. The present study sought to provide rigorous evidence by conducting a cluster-randomized trial with 1,530 undergraduates in 96 lab sections across three different introductory computer science courses. Contrary to expectations, pair programming is unrelated to virtually all short-term and longer-term outcomes among all students. Subgroup analyses indicate that pair programming leads to poorer outcomes among White students, including a lower likelihood of majoring or minoring in computer science and of completing or attempting several subsequent computer science courses. No significant findings are observed for Students of Color, and the patterns are also generally consistent and non-significant for female and male students.

Rapid advances in computer science and technology have had a broad and lasting impact across society. In an era of smart phones, 3D printers, and global connectivity, the common good depends on continued advances in computer science and, more importantly, an educated workforce well-versed in computing's capabilities and limitations. Scholars from various disciplines increasingly assert that exposure to computational thinking and literacy should be a component of every well-rounded college graduate's education (Hambrusch, Hoffmann, Korb, Haugan, & Hosking, 2009; Vee, 2013; Wing, 2008; Yadav, Zhou, Mayfield, Hambrusch, & Korb, 2011). Students appear to agree about the importance of computer science, as these programs have seen rapid increases in enrollments in recent years (Desjardins, 2015; Jacobs, 2014; Singer, 2019).

Unfortunately, larger enrollments only exacerbate the challenges inherent in introductory computer science courses, especially in helping students acquire the programming skills necessary to pursue further coursework and eventual careers in the field. Traditionally, once the programming language has been introduced, skills are developed through a succession of increasingly challenging assignments. Because students enter these courses with a broad range of experiences, students with little or no prior exposure often struggle, leading to high attrition and loss of student interest (Biggers, Brauer, & Yilmaz, 2008; Desjardins, 2015). This dynamic is compounded among underrepresented groups; computer science, like other STEM fields, has struggled to increase the participation of women and Students of Color (Beede et al., 2011; Cheryan, Ziegler, Montoya, & Jiang, 2017; Estrada et al., 2016).

Active learning strategies offer the potential to address some of these issues (Bowman & Culver, 2018; Freeman et al., 2014). In particular, pair programming (in which two students program collaboratively) has been offered as a means to increase student learning, grades, confidence, persistence, and satisfaction, but the actual causal impact of this pedagogical approach is unclear. This study brings rigorous evidence to bear on the outcomes of implementing pair programming in the computer science classroom by attempting to untangle contradictory claims and evaluate whether pair programming delivers on its promise to improve student learning and retention in computer science.

Literature on Pair Programming

Pair programming was popularized in the early 2000s by proponents of agile programming methods, a set of practices intended to build collaboration, creativity, and self-sufficiency in software development teams (Beck et al., 2001). Although pair programming is only one of the included agile methods, it is arguably the most studied and perhaps the most controversial. In its ideal form, pair programming is a practice in which two programmers work side-by-side at the same computer; they periodically switch roles between “driving” (producing code) and “navigating” (reviewing and making suggestions) (Williams & Upchurch, 2001). This practice has been increasingly adopted in both industry and classroom settings to improve efficiency, quality, and satisfaction (for reviews, see Dyba, Arisholm, Sjoberg, Hannay, & Shull, 2007; Faja, 2011; Hanks, Fitzgerald, McCauley, Murphy, & Zander, 2011; Salleh, Mendes, & Grundy, 2011; Umapathy & Ritzhaupt, 2017).

Proponents of the method cite myriad benefits for participants and instructors. For example, several scholars have observed an increase in productivity, efficiency, and coding output (Hannay, Dybå, Arisholm, & Sjøberg, 2009; Kuppuswami & Vivekanandan, 2004; Lui & Chan, 2006; Zacharis, 2011) as well as cleaner code quality with fewer bugs (Begel & Nagappan, 2008; Bipp, Lepper, & Schmedding, 2008; Cliburn, 2003; Dyba et al., 2007; Hanks et al., 2011; Hannay et al., 2009; Kuppuswami & Vivekanandan, 2004; Zacharis, 2011) and increased understanding of programming concepts (Begel & Nagappan, 2008; Faja, 2011; Hanks et al., 2011; Howard, 2006). Others have found that pair programming is positively related to persistence in the course (McDowell, Werner, Bullock, & Fernald, 2006) as well as grades on tests, assignments, and the overall course (Chigona & Pollock, 2008; Kuppuswami & Vivekanandan, 2004; McDowell et al., 2006; Mendes, Al-Fakhri, & Luxton-Reilly, 2006; Umapathy & Ritzhaupt, 2017; Wiebe et al., 2003). One group of researchers observed that students who were paired turned in their homework at significantly higher rates (Hanks, McDowell, Draper, & Krnjajic, 2004), and several scholars have found that students in pair programming courses may be more likely to declare or stay in a computer science major (Hanks et al., 2011; McDowell et al., 2006; Nagappan et al., 2003; Umapathy & Ritzhaupt, 2017).

In addition to these academic outcomes, many positive social and affective outcomes have been attributed to pair programming. This practice can enhance communication and team-building skills (Faja, 2011) as well as student confidence (Faja, 2011; Hanks et al., 2004, 2011; Lai & Xin, 2011; McDowell et al., 2006; Nosek, 1998; VanDeGrift, 2004). Numerous studies suggest that individuals enjoy programming collaboratively more than working alone (Balijepally, Mahapatra, Nerur, & Price, 2009; Bipp et al., 2008; Cao & Xu, 2005; Cliburn, 2003; Faja, 2011; Hanks, 2006; Hanks et al., 2004; Howard, 2006; McDowell et al., 2006; Melnik & Maurer, 2005; Nagappan et al., 2003; Nosek, 1998; Thomas, Ratcliffe, & Robertson, 2003; VanDeGrift, 2004; Wiebe et al., 2003). Instructors and scholars have commented that students in courses utilizing a pair programming approach are more engaged and self-sufficient (Howard, 2006; Kuppuswami & Vivekanandan, 2004; Nagappan et al., 2003; Wiebe et al., 2003), freeing up time for instructors to help struggling students (Cliburn, 2003; Nagappan et al., 2003).

However, the findings on outcomes of pair programming are not always consistent. For instance, Ally, Daroch and Toleman (2005) reported that programming professionals perceived pair programming to be less efficient than solo work in most instances, and several scholars have reported observed a loss in effort or efficiency, although this relationship was sometimes small (Bipp et al., 2008; Dyba et al., 2007). Similarly, mixed results have been reported on code quality measures (Hanks et al., 2004), and another study found that pair performance on measures such as quality is only improved for the weaker member in a pair (Balijepally et al., 2009). Others found no evidence that pair programming increased learning and understanding of concepts (Chigona & Pollock, 2008; Hanks et al., 2004), and no effect on exam scores after controlling for prior achievement (Nagappan et al., 2003).

Additionally, several scholars have commented on the challenges with implementing pair programming effectively. From a practical perspective, several instructors note the difficulty of scheduling if pairs need to meet outside class (Bevan, Werner, & McDowell, 2002; Howard, 2006; VanDeGrift, 2004). Moreover, not all studies have found that pair programming is enjoyable. For instance, Mendes et al. (2006) observed that only about half of participants wanted to experience pair programming in a future course; another study found that pair programming was unpopular and lowered morale among experienced professionals (Ally et al., 2005). Some of these differences in the enjoyment of pair programming may be predictable. For instance, more experienced and confident coders seem to be less enthusiastic about pair programming (Layman, 2006; Thomas et al., 2003), and students who are introverted or reflective also may enjoy the experience less than others (Layman, 2006). At times, the workload is not equitably shared (Nagappan et al., 2003), and differences in work ethic among paired students creates conflict (Williams, Layman, Osborne, & Katira, 2006). Indeed, pair incompatibility appears to pose the biggest threat to the effectiveness of pair programming (Ally et al., 2005; Begel & Nagappan, 2008; Bevan et al., 2002; Cao & Xu, 2005; Chaparro, Yuksel, Romero, & Bryant, 2005; Cliburn, 2003; Hanks, 2006; Nagappan et al., 2003; Van Toll III, Lee, & Ahlswede, 2007; VanDeGrift, 2004; Wiebe et al., 2003); this incomparability has been operationalized in terms of mismatch of personality and/or prior experience. However, not every study finds that mismatch is problematic, as scholars have occasionally found that heterogeneous personality pairs outperformed pairs with similar personality types (Choi, Deek, & Im, 2008; Sfetsos, Stamelos, Angelis, & Deligiannis, 2009).

There are several possible explanations for the wide range of findings. First, some significant methodological problems have been present in this research. Specifically, many previous studies suffered from small sample sizes that often did not have the statistical power to identify significant differences; these study designs also frequently failed to account for selection effects, including a lack of important control variables. Taken as a whole, these issues cast doubt on the validity of many previous findings. Second, one meta-analysis of the pair programming literature did find signs of publication bias (Hannay et al., 2009), thereby indicating that nonsignificant or negative findings have likely been underreported. Third, disparate outcomes may also be due to differences in participants, tasks, and environments (Bryant, 2004): Many of these studies utilized different methods of pairing students, were employed in different types of classrooms, were conducted for different lengths of time, were applied to different types of coding tasks, included different types of participants, and were applied in very different institutional and national contexts. Finally, pair programming may be implemented in a variety of different ways that can notably affect the results (Coman, Robillard, Sillitti, & Succi, 2014); this possibility is explored in more detail below.

Theoretical and Conceptual Framework for Equity in Computer Science

This study draws from a view of college experiences and outcomes shaped by the Diverse Learning Environments (DLE) model (Hurtado, Alvarez, Guillermo-Wann, Cuellar, & Arellano, 2012), which delineates the role of curricular and co-curricular contexts in shaping college student learning, achievement, and retention for diverse students. Experiences with coursework are situated within a context shaped by the composition of students in those environments, historical legacy of inclusion/exclusion, and psychological and behavioral dimensions of the climate. Within this broader context, the curriculum is experienced as a function of pedagogy and teaching methods, course content, instructor identities, and student identities.

For women and racially marginalized students, stigmatizing experiences and discrimination—both overt and subtle—are a common experience on college campuses (Chang, Milem, & antonio, 2011). The relationship between institutional climate and students' racial and gender identities may become more salient in STEM fields like computer science (Malcom & Feder, 2016), as issues of underrepresentation are more pronounced here than in other disciplines (Cheryan, Ziegler, Montoya, & Jiang, 2017; Estrada et al., 2016). Additionally, scholars point to the ways in which STEM fields often reflects the language and norms of White, middle-class, and masculine discourse (Cheryan, Plaut, Davies, & Steele, 2009; Lemke, 2001). Given this climate, it is perhaps unsurprising that women and Students of Color are less likely to identify with these fields (Hazari, Sadler, & Sonnert, 2013).

While women remain underrepresented in nearly all STEM fields, their participation is lowest in computer science (Cheryan et al., 2017). This underrepresentation has substantially intensified over time, and women now only comprise only about 15% of all majors (Sax et al., 2017). A wealth of scholarship has examined and attempted to address this pervasive issue, but the problem has stubbornly persisted, especially in Western cultures (Vitores & Gil-Juárez, 2016). While many of the explanations offered focus on gendered differences in prior experience, behavior, and interest (e.g. Beyer, 2014; Cheryan et al., 2017; Krieger, Allen, & Rawn, 2015; Sinclair & Kalvala, 2015), other explanations highlight the pervasive masculine culture and hostile learning environments found in computer science programs (Cheryan et al., 2017). Beyer (2014) found that classroom environment and instruction are strongly related to students' decisions to continue in a computer science major. Complementing this finding, a review of published studies on computer-supported collaborative learning found that gender differences were reduced or even non-existent when participation and inclusion were promoted explicitly, suggesting that pedagogy matters a great deal in achieving equitable outcomes by gender (Prinsen, Volman, & Terwel, 2007).

Likewise, Students of Color—specifically Black, Latinx, and Indigenous students—remain underrepresented in computer science programs (Malcom & Feder, 2016; also see Museus, Palmer, Davis, & Maramba, 2011), but this issue is less studied than gender composition. Racial underrepresentation is likely due in part to microaggressions and bias encountered in the learning environment, which Students of Color are more likely to experience (Barker, McDowell, & Kalahar, 2009). Some research has also linked a field's pervasive beliefs about innate brilliance and ability with levels of underrepresentation in that field (Leslie, Cimpian, Meyer, & Freeland, 2015), perhaps because this belief underscores classroom practices that are highly competitive and do not promote active participation. In such settings, underrepresentation may lead to tokenization and heightened pressure (Malcom & Feder, 2016) or stereotype threat (Steele, 1997).

Active and collaborative learning strategies have been proposed as a promising way to foster a more inclusive environment for marginalized students in STEM classrooms (Varma, 2006). These strategies can potentially benefit all students; a meta-analysis of 225 studies on student performance in undergraduate STEM coursework shows that active learning increases student performance in STEM classes, with particularly larger effects in smaller class sizes (Freeman et al., 2014). The best available evidence further suggests that active learning strategies may be most effective at bolstering learning and content mastery among students who have traditionally been underrepresented (for a review, see Bowman & Culver, 2018). In one study, a lecture of an introductory biology course was replaced with an interactive workshop. This introduction of active learning benefited all students in terms of exam performance and grades, but women and Students of Color had larger increases than men and White students, respectively (Preszler, 2009). Another intervention compared team-based learning, which consists of interactive small-group sessions, with PowerPoint lectures within a veterinary course; similarly, female students benefited more from the team-based learning approach than did male students (Malone & Spieth, 2012).

A fair portion of the research over the last two decades supports claims that collaborative paired work is an active learning strategy that is particularly effective in computer science classrooms (Braught, Wahls, & Eby, 2011). While active learning is seen as widely beneficial for all students, it may be particularly beneficial for students who are marginalized in the learning space. In the case of pair programming, the collaborative approach to learning may counter isolation and competition in the classroom. Engaging in meaningful interactions across difference is a highly effective approach for improving intergroup attitudes within and outside of the classroom (Allport, 1954/1985; Bowman, 2011; Crisp & Turner, 2011; Denson, 2009;

Pettigrew, 1998), so pair programming may help to improve the climate by breaking down intergroup barriers and challenging stereotypes.

That said, the research on pair programming has yielded some inconsistent results, suggesting that there is more to understand in employing this as a pedagogical tool in the classroom. One possible explanation for the mixed findings is that pair programming often diverges from the ideal structure in practice, which can then adversely affect the outcomes of this practice (Coman et al., 2014; Wiebe et al., 2003; Williams & Kessler, 2002). For instance, one study observed less clear role differentiation than proposed by proponents of pair programming (Bryant, Romero, & du Boulay, 2008). Somewhat conversely, Howard (2006) observed that she had to actively encourage students to switch roles, but they did get better at this with time and practice. Other research indicates that such dominating behavior is more likely when there is a mismatch in gender (Williams, Wiebe, Yang, Ferzli, & Miller, 2002) or in achievement and skill (Chaparro et al., 2005; Hanks et al., 2011; Stephens & Rosenberg, 2003; Williams et al., 2002); this monopolizing then results in disengagement for the other partner. Another study found that interruptions, tasks that are too simple, social pressure to avoid looking ignorant, and time constraint can all lead to disengagement in pairs (Plonka, Sharp, & van der Linden, 2012).

Present Study

This study sought to address the challenges of previous research by performing a rigorous multi-year examination of the impact of pair programming. Specifically, we conducted a cluster-randomized trial that assigned lab sections to engage in either pair programming or traditional programming; this experimental design leads to much stronger causal inferences than in previous work. We also implemented the pair programming phase of this study over four semesters within three different computer science courses that had several lead instructors and dozens of teaching assistants, which helps promote the generalizability of the findings across contexts. The sample consisted of over 1,500 undergraduate students and nearly 100 discussion sections, so the analyses had sufficient statistical power to detect main effects among all students and separately by course and by demographics. In fact, this sample contained approximately 600 Students of Color and 600 female students, which provided a unique opportunity to conduct large-scale subgroup analyses. Finally, we collected data well after the initial course enrollment so that we could explore longer-term outcomes of subsequent computer science course taking, academic success in those courses, and having a subsequent major or minor in computer science.

Method

Study Context, Procedure, and Participants

This study included undergraduates who took an introductory computer science at a large, Midwestern research university from Fall 2016 to Spring 2018. The institution offers three introductory courses: a class designed for computer science majors (i.e., CS 1), a class designed for humanities majors (i.e., CS 0), and a class designed for social and information science majors (i.e., CS ½). These courses vary considerably in the amount of technical knowledge and skills required for assignments and exams. The format of all three courses consisted of large lecture classes that met either twice per week (CS 0 and CS ½) or three times per week (CS 1) as well as one lab section per week (50 minutes for all courses). A full-time instructor led the lectures, and graduate teaching assistants (TAs) led the lab sections. Each TA facilitated either two lab sections (for CS 1) or three sections (for CS 0 and CS ½) per semester.

Lab sections were randomly assigned to engage in pair programming or in individual programming practices. Aside from this use of collaborative learning in some sections, the rest of the lab section was identical, including the assignments that students were required to complete.

To minimize any potential effects of TA skill, the random assignment occurred within TA, such that each TA led at least one paired section and at least one individual section. A total of 96 lab sections (49 paired and 47 individual) were offered during the two years of data collection. The researchers and primary course instructors trained the TAs on how to facilitate pair programming within the lab sections. Students in the experimental condition participated in three different pairings throughout the semester; this frequency was intended to give students enough time to become comfortable with their partner and also to allow students to engage with different partners.

With the exception of one course in the first semester of the study that allowed some work outside of class, all pair programming was conducted during lab sections (scheduling difficulties for students meeting outside of class prevented this option from being viable). Students in paired sections were instructed to use pair programming in a method consistent with formal definitions of this practice. One student would start as the driver, taking lead on using the keyboard and writing code. The other student would act as the navigator, reviewing code, looking for errors, and making suggestions while keeping the bigger goals of the assignment in focus. After a period of time, students were supposed to switch roles, continuing in this fashion until the assignment was complete. Students were also given a handout with tips and strategies for working effectively on this type of collaborative task.

Participants were included in the present analytic sample if they (a) were enrolled in one of the introductory computer science courses after the fourth week of the semester, (b) were in their first semester of one of these courses (some students enrolled multiple times within the same course and/or enrolled in more than one of these courses), (c) were an undergraduate student (rather than a graduate or non-degree-seeking student), (d) gave permission for their course data to be linked with their registrar data, and (e) had available data on coursework after the experimental semester (some students took this class in their last semester at the university). This sample included 1,530 undergraduates; 39% were female, 26% were first-generation students (neither parent had attended postsecondary education), 60% were White/Caucasian, 20% were international students (the vast majority of whom were from Asian countries), 7% were Latinx/Hispanic, 5% were Asian American, 4% were Black/African American, and 4% were multiracial or from another racial group. Moreover, 30% of participants were in their first year, 26% were in their second year, 25% were in their third year, and 18% were in their fourth year or beyond. Twenty-seven percent of students were computer science majors when they were enrolled in the introductory course.

Within this sample, 796 students were enrolled in one of the 49 lab sections that used pair programming, whereas 734 students were enrolled in one of the 47 lab sections that used individual programming. Randomization appears to have yielded equivalent treatment and control groups, since no significant differences were observed across experimental condition in terms of course, semester enrolled, year in college, race, sex, first-generation status, age, veteran status, U.S. citizenship, in-state residency, composite ACT score, combined SAT score, and high school GPA, *p*s $\geq$ .10.

Measures

Two of the primary outcome variables indicated whether students had a major or a minor in computer science. Registrar data was obtained from every semester in the academic year from Fall 2016 to Spring 2019; this use of repeated data pulls was necessary to obtain the best possible indication of students' major and minor, since some students graduated, transferred, or dropped out before the Spring 2019 semester. We used Spring 2019 data whenever possible; for students who were not enrolled in that semester, we used the most recent term in which they attended the university. In addition to these measures, several variables were used to describe students' success within their introductory computer science course. These outcomes included the letter grade in the course (A+ = 4.33 to F = 0), proportion of total points received in the course, average proportion of points received on all exams, and grade received on the final exam. The letter grade and total proportion of points were both considered here, since the grades within the courses were curved and assigning letter grades limited the distribution, so the results may differ across these two measures.

Other outcomes focused on students' subsequent computer science coursework. We obtained data on all computer science courses that students completed (with a passing grade) or attempted (and received any grade, including an F or W). This information was used to compute variables for the total number of computer science courses completed and the total number attempted. Several binary indicators were used to measure students' enrollment in several key courses: Discrete Structures and Data Structures (both of these are required for computer science majors and are prerequisites for taking various other courses in the major); Programming for Informatics (this course is required for informatics majors, but not computer science majors); and CS 1 (this outcome was only examined among students who were enrolled in CS 0 or CS ½ during their treatment semester). Three variables were created for each course: whether it was completed successfully (0 = no, 1 = yes), whether it was attempted (0 = no, 1 = yes), and the grade that students earned (A+ = 4.33 to F = 0).

The primary independent variable was whether students participated in a lab section that used pair programming (0 = no, 1 = yes). Dummy variables were used to indicate the course in which they were enrolled during the experiment (CS ½ and CS 1, with CS 0 as the referent group) and the semester in which they participated (Spring 2017, Fall 2017, and Spring 2018, with Fall 2016 as the referent group). The moderators of interest were students' race/ethnicity (0 = Student of Color, 1 = White/Caucasian) and sex (0 = male, 1 = female). Descriptive statistics for all variables appear in the Appendix.

Analyses

Because students were nested within lab sections, hierarchical linear modeling (HLM) analyses were conducted (see Raudenbush & Bryk, 2002; Snijders & Bosker, 2012). Lab sections were also nested within courses; however, there were only three introductory courses, so course was modeled via dummy variables. Semester was also included in all analyses to account for any differences over time. Supplemental analyses showed that the results reported here were substantively identical if these data were analyzed using regression analyses with cluster-robust standard errors.

Many of the outcomes were binary in nature (i.e., computer science major and minor, enrollment in individual subsequent courses, successful completion of each of those courses). These outcomes were appropriately treated as binary using hierarchical generalized linear modeling analyses. Not surprisingly, the total number of subsequent computer science courses completed and attempted were both highly skewed, as they contained a large number of students who completed no courses after the treatment semester. Therefore, zero-inflated negative binomial regression analyses were conducted; these analyses conducted separate tests to predict whether participants engaged in zero courses (versus at least one course) and to predict the number of courses in which they enrolled (among students who completed at least one course; see Hoffmann, 2016; Long, 1997). Finally, several outcomes were treated as continuous via traditional hierarchical linear modeling analyses: grades in subsequent courses and the outcomes within the treatment course (final grade as well as the proportion of total points received overall, on all exams, and on the final exam).

Some analyses examined interactions between pair programming and one of the demographic characteristics (race or sex). To model these interactions appropriately, the multilevel analyses contained variables for pair programming, the demographic attribute, and the multiplicative interaction between these two variables (Hayes, 2018; Jaccard & Turrisi, 2003). Subgroup analyses were also conducted to explore the impact of pair programming within each course and within racial/ethnic groups.

Limitations

The most notable limitation of this study is that it examined pair programming at a single institution. We sought to bolster generalizability as much as possible within this constraint by examining three different courses (with modestly different implementations of pair programming) and exploring potential moderation effects across student demographics, but the results may not apply to other institutions. Later in the paper, we discuss a couple of key features of this university's approach to pair programming that may have affected the findings. Furthermore, although we were able to examine various post-course outcomes, a longer time period would have allowed us to explore graduation outcomes, including whether students graduated at all and whether they did so with a computer science major or minor.

Results

Main Effects Overall and by Course

The findings for the effects of pair programming on having a computer science major or minor appear in Table 1. Within the full sample, students who participated in pair programming are significantly *less* likely to be computer science majors than those who participated in individual programming; this difference across experimental conditions was 5.2 percentage points. When examining each course separately, this same significant pattern is observed for students who originally enrolled in CS 1, such that students in pair programming are 7.6 percentage points less likely to be computer science majors than are students in individual programming. No other effects are significant for other courses predicting computer science majors or for the analyses predicting computer science minors overall and by course.

Table 2 displays the results for pair programming predicting participation and success in computer science coursework after the treatment semester. These analyses were conducted among all students and separately for subsequent courses that follow logically within the curriculum (e.g., informatics majors are required to take both CS ½ and Programming for Informatics, so subgroup analyses were conducted predicting this outcome for those who took CS ½). Only two of the 39 total analyses were significant at a threshold of $p < .05$, which is consistent with what one might expect to find via random chance. These two results were also significant in opposite directions: Pair programming was negatively associated with completing the Discrete Structures course among all students, whereas it was positively associated with the number of courses attempted among students who initially took CS 1 (and who took at least one course). That said, some potential patterns emerge when using a more lenient threshold for statistical significance ($p < .10$): pair programming predicts a lower likelihood of completing or attempting Discrete Structures among all students and those who started in CS 1, whereas pair programming is positively related to completing and attempting the Programming for Informatics course among students who originally took CS ½. At this threshold of same significance, pair programming is also positively related to the total number of computer science courses completed among CS 1 students who took at least one computer science course and negatively related to attempting CS 1 among CS ½ students.

The results for grades that occurred during the treatment course appear in Table 3. Pair programming is inversely related to the final course grade among all students (about .13 of a grade point) and to the percentage correct on the final exam among students in CS ½ (about six percentage points). When using a more lenient threshold for significance ($p < .10$), pair programming also leads to a lower final course grade and lower percentage of total points received among CS ½ students (about ¼ of a grade point and three percentage points, respectively). No other relationships for these treatment course grades are significant.

Moderation and Subgroup Analyses by Student Characteristics

The impact of pair programming clearly differs as a function of students' race. As shown in Table 4, pair programming contributes to various negative outcomes among White students, including a lower likelihood of having a computer science major or minor as well as completing or attempting Discrete Structures, Data Structures, and CS 1 (the latter analyses occurred only for those who initially took CS 0 or ½). Pair programming also led to a lower final course grade in the treatment semester among White students. Conversely, pair programming was not significantly related to any outcome among Students of Color. These divergent results were not simply the function of disparate sample sizes, as Students of Color comprised 40% of all participants. In addition, treatment x race interactions within the full sample were significant for several outcomes at $p < .05$ (computer science minor as well as completing and attempting Data Structures) and for additional outcomes at $p < .10$ (completing Discrete Structures as well as completing and attempting CS 1).

Because the vast majority of the outcomes presented in Table 4 are binary, it can be difficult to interpret the magnitude of these effects using traditional log-odds coefficients. Therefore, Table 5 provides the means separately by experimental condition among White students, along with two effect size metrics of these disparities. When examining all White students, many of these differences fall near effect size guidelines for college impact studies of small relationships (five percentage points) or between small and medium (nine percentage points; see Mayhew, Rockenbach, Bowman, Seifert, & Wolniak, 2016). For example, White students in the individual programming condition are 7.2 percentage points more likely to major in computer science than those in the pair programming condition. These percentage-point effect sizes are generally larger when limiting the sample to students in the most relevant course (i.e., the one that was most likely to lead to the corresponding outcome). For instance, the effect of individual programming (versus paired) on completing Discrete Structures increases from 6.6 percentage points among all White students to 11.2 percentage points among White students who took CS 1 during the treatment semester. The largest effect size occurs for attempting CS 1 among students who originally took CS ½; this 15 percentage-point increase is equal to Mayhew et al.'s guideline for a large effect. The lone exception to these favorable patterns for individual programming among White students is that pair programming leads to modestly higher rates of completing and attempting the Programming for Informatics course.

The effect size for group differences can also be indicated via the percentage difference in the probability of achieving a binary outcome; this metric may sometimes be at least as useful as the percentage-point difference. For instance, individual programming resulted in a 2.3 percentage-point increase in White students' having a minor in computer science, which might seem like a trivial amount. However, given the small percentage of students who enroll in that minor, White students in the individual condition are more than twice as likely as those in the paired condition to do so (135% increase among all students and 148% increase among students in CS 1). The percentage increases were far more modest for the other outcomes (also shown in Table 5). For instance, individual programming led to a 35% increase in computer science majors among all White students, along with a 17% increase among those who enrolled in CS 1 during the treatment semester. Aside from minoring in computer science, the largest percentage increase occurs for attempting CS 1 among White students who started in CS 0 or CS ½ (60%) and solely those who started in CS ½ (74%).

In contrast to the pronounced findings by race, significant differences rarely emerged for the other two student characteristics of interest. Across the various analyses, the only significant treatment x gender interaction at $p < .05$ was that pair programming is more positively related to attempting CS 1 among female students than among male students. Three other interactions are significant at $p < .10$ (the impact of pair programming for female students is more positive for completing CS 1 and grade within the treatment course, whereas the reverse is true for the total number of computer science courses). Subgroup analyses by sex showed almost no significant main effects, except for a couple of negative findings for pair programming among male students. Therefore, it appears that the impact of pair programming does not vary systematically by students' sex.

Discussion

This cluster-randomized trial found that pair programming has no short-term or long-term benefits in terms of students' grades or their future engagement with computer science coursework and degree programs. If anything, pair programming contributed to a modest reduction in grades within the treatment course and in the subsequent likelihood of majoring in computer science among all students. Furthermore, the findings were frequently negative among

White students, as pair programming contributed to non-trivial reductions in subsequent participation in computer science courses, minor, and major. The present results contrast notably with research that generally obtains positive results for pair programming within both educational and workplace settings (although substantial heterogeneity does exist; for systematic reviews, see Dyba, Arisholm, Sjoberg, Hannay, & Shull, 2007; Faja, 2011; Hanks, Fitzgerald, McCauley, Murphy, & Zander, 2011; Salleh, Mendes, & Grundy, 2011; Umapathy & Ritzhaupt, 2017).

Why do the findings of the present study diverge from prior research? Several factors may have contributed individually or in combination. First, the use of a cluster-randomized trial facilitates strong conclusions about the causal effects of pair programming, whereas selection bias may have been a notable problem in much of the prior research. Students who were concerned that they were not adequately prepared or felt that pair programming would be too much work might have decided to avoid coursework that required students to work in pairs. Second, Hannay et al.'s (2009) meta-analysis found that publication bias was a notable concern; therefore, additional papers that obtained nonsignificant or negative findings for pair programming likely exist, but these have not been published. Third, a surprising number of pair programming studies do not test for statistical significance at all, and those that conduct formal tests tend to omit key independent variables that may explain a spurious relationship between pair programming and the outcome(s) of interest. Thus, the prior research may overstate the actual relationship above and beyond the role of publication bias.

Fourth, the implementation of pair programming in the present study may have been less than ideal. The lab sections for all three courses were only 50 minutes long, and virtually all of the pair programming only took place within the lab section (with the exception of one semester and one course mentioned earlier). This reasonably short amount of class time may not have been enough for students to engage with this collaborative learning practice in sufficient depth. As another potentially related issue, students in the treatment condition participated in three different pairings throughout the semester. Switching partners was intended to provide students with new opportunities to engage with other class members (and with a built-in timeline for leaving a partner who they might not like), but these changes may have also made it more difficult for students to adjust to pair programming effectively.

That said, the subjective experiences of students and TAs within these pair programming sections were mostly positive (Jarratt, Bowman, Culver, & Segre, 2019). According to a student survey at the end of the semester, 50% of students who engaged in pair programming would probably or definitely recommend this practice, whereas only 25% would probably or definitely not do so; these perceptions are similar to those in Mendes et al. (2006). Moreover, the TAs generally reported a more positive classroom environment (in terms of student engagement, effort, and class attendance) in the paired sections than the individual sections. When asked whether they would use pair programming in the future, 68% of TAs said yes, and only 11% said no. These findings suggest that pair programming may have been implemented in a reasonably appropriate manner within the present study.

It is also worth acknowledging differences that seem unlikely to explain the disparate findings between this study and previous work. Most pair programming research in college contexts has examined coursework for computer science majors, whereas this study includes coursework for majors and non-majors, including students in CS 0 who are extremely unlikely to major in computer science. However, the negative findings in the present study are actually more prevalent within CS 1 than in CS ½ or CS 0, so this broader sampling does not appear to explain the divergence. This study also uses a wide array of outcomes (including post-course measures that are collected later than in most previous work), but the findings across both short-term and longer-term outcomes are non-significant at best. The potential lone possible instance in which pair programming has a benefit in the present study occurs for taking the Programming for Informatics course, which is not required for computer science majors (but is required for informatics majors).

The fact that pair programming exhibited various negative outcomes among White students, but not among Students of Color, is quite notable. Numerous negative results occurred for participation in degree programs (computer science major and minor) and in future coursework (completing and attempting several subsequent classes). However, almost no negative findings are evident for course grades; the only exception was that White students in pair programming performed worse on their final grade in the treatment course than those in individual programming, but no significant differences occurred for exams within that course. Taken together, these patterns suggest that pair programming affects White students' participation in computer science coursework rather than their achievement or knowledge within the courses that they do take.

The reasons for this racialized pattern are unclear. Some prior research has found that active and collaborative learning provides benefits for all students that are even more positive among Students of Color and students from other minoritized backgrounds (see Bowman & Culver, 2018); the present results exhibit this same relative difference across groups, but without the presence of positive effects for any group. The negative results for White students may stem, at least in part, from the fact that the computer science classrooms are more racially diverse than the undergraduate population of the university and the surrounding community, especially in terms of the number of Asian and Asian American students. Previous research has shown that situational cues regarding demographic representation (or lack thereof) can affect college students' sense of belonging and interest in engaging within STEM contexts (Murphy, Steele, & Gross, 2007). The effects of situational cues and social identity threats generally occur among groups that are negatively stereotyped, but computer science is a domain in which Asians and Asian Americans may be viewed as the model students (e.g., McGee, Thakore, & LaBlance, 2017). If White students perceive this stereotype, then interacting frequently with Students of Color via pair programming may have heightened the threat that White students could experience within these environments. Supplemental analyses showed that White students and Students of Color in this sample did not differ in their prior programming experience; observing this equality via engagement in pair programming would further challenge the presumption of superiority that White students may hold in other academic contexts (see Walton & Cohen, 2003).

Conclusion

This rigorous study provides intriguing results about how pair programming may not bolster desired outcomes and may actually reduce future participation in computer science. This pattern is surprising, since a substantial and robust literature supports the positive effects of active and collaborative learning more broadly on various academic outcomes (e.g., Freeman et al., 2014; Kyndt et al. 2013). However, this pedagogical approach may not result in its intended effective implementation of collaborative practices. Pair programming provides the possibility for fruitful engagement within pairs, but its structure (with only one person at the computer at a time) may also lead to one partner doing the vast majority of the work, while the other partner may not contribute much to the assignment or learn much as a result (Chaparro et al., 2005;

Hanks et al., 2011; Howard, 2006; Nagappan et al., 2003; Stephens & Rosenberg, 2003; Williams et al., 2002). Pairing a more experienced student with a less experienced student can lead to a mentoring relationship in ideal circumstances, but it can also be intimidating and disengaging. Indeed, being randomly assigned an experienced partner (rather than an inexperienced partner) in pair programming may lead to adverse outcomes, including less time in the driving role, poorer effort on the assignment, weaker understanding of relevant course concepts, and even lower overall interest in computer science (Bowman, Jarratt, Culver, & Segre, 2019).

Although a fair number of studies have already examined the effects of pair programming on college student outcomes, additional research that facilitates strong causal inferences is needed. The preceding section of this paper discussed several reasons why this paper may have provided different results than previous work on pair programming, and a very important issue is the prevalence of flawed studies, including those that are correlational in nature. By employing stronger research designs, future inquiry will be able to focus on uncovering the conditions in which pair programming may be effective or ineffective. Given problems with publication bias in prior research (Hannay et al., 2009), understanding whether and when pair programming does not promote desired outcomes—rather than only publishing work that yields favorable results—is critical for advancing knowledge.

When considering implications for practice, one could argue for the use of pair programming even if it had no effect on student outcomes. This collaborative approach encourages students to ask each other questions and solve problems as a team rather than frequently taking instructors' time to ask for help (Cliburn, 2003; Howard, 2006; Jarratt et al., 2019; Kuppuswami & Vivekanandan, 2004; Nagappan et al., 2003). Collaborative learning can also foster interpersonal interactions across difference that are associated with a wide variety of educational benefits (see Mayhew et al., 2016). As a practical consideration, university computer labs would only need half the number of workstations that have relevant software to accommodate students in pair programming environments. However, if pair programming has negative effects on computer science outcomes, then this practice certainly needs to be reconsidered. Future research is crucial for making this determination.

References

Allport, G. W. (1954/1985). *The nature of prejudice*. Reading, MA: Addison-Wesley.

Ally, M., Darroch, F., & Toleman, M. (2005). A framework for understanding the factors influencing pair programming success. In H. Baumeister, M. Marchesi, & M. Holcombe (Eds.), *Extreme Programming and Agile Processes in Software Engineering* (Vol. 3556, pp. 82–91). Berlin, Heidelberg: Springer.

Balijepally V.G., Mahapatra, R.K., Nerur, S., & Price, K.H. (2009). Are two heads better than one for software development? The productivity paradox of pair programming. *MIS Quarterly*, *33*(1), 91–118.

Barker, L. J., McDowell, C., & Kalahar, K. (2009). Exploring factors that influence computer science introductory course students to persist in the major. *ACM SIGCSE Bulletin, 41*(1), 153–157.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., … Thomas, D. (2001). Manifesto for agile software development. Retrieved from http://www.agilemanifesto.org

Beede, D. N., Julian, T. A., Langdon, D., McKittrick, G., Khan, B., & Doms, M. E. (2011). Women in STEM: A gender gap to innovation. *SSRN Electronic Journal*.

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

Bevan, J., Werner, L., & McDowell, C. (2002). Guidelines for the use of pair programming in a freshman programming class (pp. 100–107). IEEE Computer Society.

Beyer, S. (2014). Why are women underrepresented in Computer Science? Gender differences in stereotypes, self-efficacy, values, and interests and predictors of future CS course-taking and grades. *Computer Science Education*, *24*(2–3), 153–192.

Biggers, M., Brauer, A., & Yilmaz, T. (2008). Student perceptions of computer science: A retention study comparing graduating seniors vs. CS leavers. *ACM SIGCSE Bulletin - SIGCSE '08*, *40*(1), 402–406.

Bipp, T., Lepper, A., & Schmedding, D. (2008). Pair programming in software development teams – An empirical study of its benefits. *Information and Software Technology*, *50*(3), 231–240.

Bowman, N. A. (2011). Promoting participation in a diverse democracy: A meta-analysis of college diversity experiences and civic engagement. *Review of Educational Research*, *81*(1), 29–68.

Bowman, N. A., & Culver, K. (2018). Promoting equity and student learning: Rigor in undergraduate academic experiences. In C. M. Campbell (Ed.), *Reframing notions of rigor: Building scaffolding for equity and student success* (New Directions for Higher Education, no. 181, pp. 47-57). San Francisco, CA: Jossey-Bass.

Bowman, N. A., Jarratt, L., Culver, K., & Segre, A. M. (2019). *How prior programming experience affects students' pair programming experiences and outcomes*. Unpublished paper, University of Iowa.

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

Bryant, S. (2004). Double trouble: Mixing qualitative and quantitative methods in the study of eXtreme programmers. In *2004 IEEE Symposium on Visual Languages - Human Centric Computing* (pp. 55–61). Rome: IEEE.

Bryant, S., Romero, P., & du Boulay, B. (2008). Pair programming and the mysterious role of the navigator. *International Journal of Human-Computer Studies*, *66*(7), 519–529.

Cao, L., & Xu, P. (2005). Activity patterns of pair programming. In *Proceedings of the 38th Hawaii International Conference on System Sciences*. Big Island, HI, USA: IEEE.

Chang, M. J., Milem, J. F., & antonio, anthony lising. (2011). Campus climate and diversity. In J. H. Schuh, S. R. Jones, S. R. Harper, & S. R. Komives (Eds.), *Student services: A handbook for the profession* (5th ed, pp. 43–58). San Francisco: Jossey-Bass.

Chaparro, E. A., Yuksel, A., Romero, P., & Bryant, S. (2005). Factors affecting the perceived effectiveness of pair programming in higher education. *PPIG*.

Cheryan, S., Plaut, V. C., Davies, P. G., & Steele, C. M. (2009). Ambient belonging: How stereotypical cues impact gender participation in computer science. *Journal of Personality and Social Psychology*, *97*(6), 1045–1060.

Cheryan, S., Ziegler, S. A., Montoya, A. K., & Jiang, L. (2017). Why are some STEM fields more gender balanced than others? *Psychological Bulletin*, *143*(1), 1–35.

Chigona, W., & Pollock, M. (2008). Pair programming for information systems students new to programming: Students’ experiences and teachers’ challenges. In *PICMET ’08 - 2008 Portland International Conference on Management of Engineering & Technology* (pp. 1587–1594). Cape Town, South Africa: IEEE.

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*(11), 1114–1126.

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

Coman, I. D., Robillard, P. N., Sillitti, A., & Succi, G. (2014). Cooperation, collaboration and pair-programming: Field studies on backup behavior. *Journal of Systems and Software, 91*, 124–134.

Crisp, R. J., & Turner, R. N. (2011). Cognitive adaptation to the experience of social and cultural diversity. *Psychological Bulletin, 137*(2), 242–266.

Denson, N. (2009). Do curricular and cocurricular diversity activities influence racial bias? A meta-analysis. *Review of Educational Research, 79*(2), 805–838.

Desjardins, M. (2015, October 22). The real reason U.S. students lag behind in computer science. *Fortune*. Retrieved from http://fortune.com/2015/10/22/u-s-students-computer-science/

Dyba, T., Arisholm, E., Sjoberg, D. I. K., Hannay, J. E., & Shull, F. (2007). Are two heads better than one? On the effectiveness of pair programming. *IEEE Software*, *24*(6), 12–15.

Estrada, M., Burnett, M., Campbell, A. G., Campbell, P. B., Denetclaw, W. F., Gutiérrez, C. G., … Zavala, M. (2016). Improving underrepresented minority student persistence in STEM. *CBE—Life Sciences Education*, *15*(3), 1-10.

Faja, S. (2011). Pair programming as a team based learning activity: A review of research. *Issues in Information Systems*, *XII*(2), 207–216.

Freeman, S., Eddy, S. L., McDonough, M., Smith, M. K., Okoroafor, N., Jordt, H., & Wenderoth, M. P. (2014). Active learning increases student performance in science, engineering, and mathematics. *Proceedings of the National Academy of Sciences*, *111*(23), 8410–8415.

Hambrusch, S., Hoffmann, C., Korb, J. T., Haugan, M., & Hosking, A. L. (2009). A Multidisciplinary Approach Towards Computational Thinking for Science Majors. *ACM SIGCSE Bulletin - SIGCSE '09*, *41*(1), 183–187.

Hanks, B. (2006). Student attitudes toward pair programming. In *Proceedings of the 11th annual SIGCSE conference on innovation and technology in computer science education* (pp. 113–117). Bologna, Italy.

Hanks, B., Fitzgerald, S., McCauley, R., Murphy, L., & Zander, C. (2011). Pair programming in education: A literature review. *Computer Science Education*, *21*(2), 135–173.

Hanks, B., McDowell, C., Draper, D., & Krnjajic, M. (2004). Program quality with pair programming in CS. In *Proceedings of the 9th annual SIGCSE conference on innovation and technology computer science education* (pp. 176–180). Leeds, UK.

Hannay, J. E., Dybå, T., Arisholm, E., & Sjøberg, D. I. K. (2009). The effectiveness of pair programming: A meta-analysis. *Information and Software Technology*, *51*(7), 1110–1122.

Hayes, A. F. (2018). *Introduction to mediation, moderation, and conditional process analysis: A regression-based approach* (2nd ed.). New York, NY: Guilford.

Hazari, Z., Sadler, P. M., & Sonnert, G. (2013). The science identity of college students: Exploring the intersection of gender, race, and ethnicity. *Journal of College Science Teaching*, *42*(5), 82–91.

Hoffmann, J. P. (2016). *Regression models for categorical, count, and related variables: An applied approach*. Oakland, CA: University of California Press.

Howard, E. V. (2006). Attitudes on using pair-programming. *Journal of Educational Technology Systems*, *35*(1), 89–103.

Hurtado, S., Alvarez, C. L., Guillermo-Wann, C., Cuellar, M., & Arellano, L. (2012). A model for diverse learning environments. In J. C. Smart & M. B. Paulsen (Eds.), *Higher Education: Handbook of Theory and Research: Volume 27* (pp. 41–122). Dordrecht: Springer Netherlands.

Jaccard, J., & Turrisi, R. (2003). *Interaction effects in multiple regression* (No. 72). Thousand Oaks, CA: SAGE Publications.

Jacobs, P. (2014, September 15). Here's why more than 800 Harvard students signed up for a notoriously hard computer science class. *Business Insider*. Retrieved from https://www.businessinsider.com/why-so-many-harvard-students-take-computer-science-2014-9

Jarratt, L., Bowman, N. A., Culver, K. & Segre, A. M. (2019). *Pair programming as a pedagogical approach for promoting success and equity in computer science*. Unpublished data, University of Iowa.

Krieger, S., Allen, M., & Rawn, C. (2015). Are females disinclined to tinker in computer science? In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education - SIGCSE '15* (pp. 102–107). Kansas City, Missouri, USA: ACM Press.

Kuppuswami, S., & Vivekanandan, K. (2004). The effects of pair programming on learning efficiency in short programming assignments. *Informatics in Education, 2*, 251-266.

Kyndt, E., Raes, E., Lismont, B., Timmers, F., Cascallar, E., & Dochy, F. (2013). A meta-analysis of the effects of face-to-face cooperative learning: Do recent studies falsify or verify earlier findings? *Educational Research Review, 10*, 133-149.

Lai, H., & Xin, W. (2011). An experimental research of the pair programming in java programming course. In *Proceeding of the International Conference on e-Education, Entertainment and e-Management* (pp. 257–260). Bali, Indonesia: IEEE.

Layman, L. (2006). Changing students perceptions: An analysis of the supplementary benefits of collaborative software development. In *19th Conference on Software Engineering Education & Training*. Turtle Bay, HI, USA.

Lemke, J. L. (2001). Articulating communities: Sociocultural perspectives on science education. *Journal of Research in Science Teaching*, *38*(3), 296–316.

Leslie, S.-J., Cimpian, A., Meyer, M., & Freeland, E. (2015). Expectations of brilliance underlie gender distributions across academic disciplines. *Science*, *347*(6219), 262–265.

Long, J. S. (1997). *Regression models for categorical and limited dependent variables*. Thousand Oaks, CA: Sage.

Lui, K. M., & Chan, K. C. C. (2006). Pair programming productivity: Novice–novice vs. expert–expert. *International Journal of Human-Computer Studies*, *64*(9), 915–925.

Malcom, S., & Feder, M. (Eds.). (2016). *Barriers and opportunities for 2-year and 4-year STEM degrees: Systemic change to support students' diverse pathways*. Washington, D.C.: National Academies Press.

Malone, E., & Spieth, A. (2012). Team-based learning in a subsection of a veterinary course as compared to standard lectures. *Journal of the Scholarship of Teaching and Learning*, *12*(3), 88–107.

Mayhew, M. J., Rockenbach, A. N., Bowman, N. A., Seifert, T. A., & Wolniak, G. C., with Pascarella, E. T., & Terenzini, P. T. (2016). *How college affects students (Vol. 3): 21st century evidence that higher education works.* San Francisco, CA: Jossey-Bass.

McDowell, C., Werner, L., Bullock, H. E., & Fernald, J. (2006). Pair programming improves student retention, confidence, and program quality. *Communications of the ACM*, *49*(8), 90–95.

McGee, E. O., Thakore, B. K., & LaBlance, S. S. (2017). The burden of being “model”: Racialized experiences of Asian STEM college students. *Journal of Diversity in Higher Education*, *10*(3), 253–270.

Melnik, G., & Maurer, F. (2005). A cross-program investigation of students’ perceptions of agile methods. In *Proceedings of the 27th International Conference on Software Engineering*. St. Louis, MO.

Mendes, E., Al-Fakhri, L. B., & 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 SIGCSE conference on innovation and technology in computer science education* (pp. 108–112). Bologna, Italy.

Murphy, M. C., Steele, C. M., & Gross, J. J. (2007). Signaling threat: How situational cues affect women in math, science, and engineering settings. *Psychological Science 18*(10), 879-885.

Museus, S. D., Palmer, R. T., Davis, R. J., & Maramba, D. C. (2011). *Racial and ethnic minority students’ success in STEM education*. San Francisco, CA: Jossey-Bass Inc.

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

Nosek, J. T. (1998). The case for collaborative programming. *Communications of the ACM*, *41*(3), 105–108.

Pettigrew, T. F. (1998). Intergroup contact theory. *Annual Review of Psychology*, *49*(1), 65–85.

Plonka, L., Sharp, H., & van der Linden, J. (2012). Disengagement in pair programming: Does it matter? In *2012 34th International Conference on Software Engineering (ICSE)* (pp. 496–506). Zurich: IEEE.

Preszler, R. W. (2009). Replacing lecture with peer-led workshops improves student learning. *CBE—Life Sciences Education*, *8*(3), 182–192.

Prinsen, F. R., Volman, M. L. L., & Terwel, J. (2007). Gender-related differences in computer-mediated communication and computer-supported collaborative learning: Gender-related differences in CMC and CSCL. *Journal of Computer Assisted Learning*, *23*(5), 393–409.

Raudenbush, S. W., & Bryk, A. S. (2002). *Hierarchical linear models: Applications and data analysis methods* (2nd ed.). Thousand Oaks, CA: Sage.

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*, *37*(4), 509–525.

Sax, L. J., Lehman, K. J., Jacobs, J. A., Kanny, M. A., Lim, G., Monje-Paulson, L., & Zimmerman, H. B. (2017). Anatomy of an enduring gender gap: The evolution of women's participation in computer science. *The Journal of Higher Education*, *88*(2), 258–293.

Sfetsos, P., Stamelos, I., Angelis, L., & Deligiannis, I. (2009). An experimental investigation of personality types impact on pair effectiveness in pair programming. *Empirical Software Engineering*, *14*(2), 187–226.

Sinclair, J., & Kalvala, S. (2015). Exploring societal factors affecting the experience and engagement of first year female computer science undergraduates. In *Proceedings of the*

15th Koli Calling Conference on Computing Education Research - Koli Calling '15 (pp. 107–116). Koli, Finland: ACM Press.

Singer, N. (2019, January 24). The hard part of computer science? Getting into class. *The New York Times*. Retrieved from https://www.nytimes.com/2019/01/24/technology/computer-science-courses-college.html

Snijders, T. A. B., & Bosker, R. (2012). *Multilevel analysis: An introduction to basic and advanced multilevel modeling* (2nd ed.). Thousand Oaks, CA: Sage.

Steele, C. M. (1997). A threat in the air: How Stereotypes Shape Intellectual Identity and Performance. *American Psychologist, 52*(6), 613–629.

Stephens, M., & Rosenberg, D. (2003). *Extreme programming refactored: The case against XP*. Berkeley, CA: Apress.

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). Reno, NV.

Umapathy, K., & Ritzhaupt, A. D. (2017). A meta-analysis of pair-programming in computer programming courses: Implications for educational practice. *ACM Transactions on Computing Education, 17*(4), 16:1-16:13.

Van Toll III, T., Lee, R., & Ahlswede, T. (2007). Evaluating the usefulness of pair programming in a classroom setting. In *6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007)* (pp. 302–308). Melbourne, Australia: IEEE.

VanDeGrift, T. (2004). Coupling pair programming and writing: Learning about students' perceptions and processes. *ACM SIGCSE Bulletin, 36*(1), 2–6.

Varma, R. (2006). Making computer science minority-friendly. *Communications of the ACM*, *49*(2), 129–134.

Vee, A. (2013). Understanding computer programming as a literacy. *Literacy in Composition Studies*, *1*(2), 42–64.

Vitores, A., & Gil-Juárez, A. (2016). The trouble with ‘women in computing’: A critical examination of the deployment of research on the gender gap in computer science. *Journal of Gender Studies*, *25*(6), 666–680.

Walton, G. M., & Cohen, G. L. (2003). Stereotype lift. *Journal of Experimental Social Psychology, 39*, 456-467.

Wiebe, E. N., Williams, L., Petlick, J., Nagappan, N., Balik, S., Miller, C., & Ferzli, M. (2003). Pair programming in introductory programming labs. In *Proceedings of the 2003 American Society for Engineering Education Annual Conference & Exposition*. American Society for Engineering Education.

Williams, L., Layman, L., Osborne, J., & Katira, N. (2006). Examining the compatibility of student pair programmers. In *AGILE 2006 (AGILE’06)* (pp. 411–420). Minneapolis, MN, USA: IEEE.

Williams, Laurie, & Upchurch, R. L. (2001). In support of student pair-programming. *ACM SIGCSE Bulletin*, *33*(1), 327–331.

Williams, Laurie, & Kessler, R. (2002). *Pair programming illuminated*. Boston: Addison-Wesley.

Williams, Laurie, Wiebe, E., Yang, K., Ferzli, M., & Miller, C. (2002). In support of pair programming in the introductory computer science course. *Computer Science Education*, *12*(3), 197–212.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *366*(1881), 3717–3725.

Yadav, A., Zhou, N., Mayfield, C., Hambrusch, S., & Korb, J. T. (2011). Introducing computational thinking in education courses. In *Proceedings of the 42nd ACM technical symposium on Computer science education - SIGCSE '11* (p. 465). Dallas, TX, USA: ACM Press.

Zacharis, N. Z. (2011). Measuring the effects of virtual pair programming in an introductory programming java course. *IEEE Transactions on Education*, *54*(1), 168–170.

Table 1. Results of multilevel analyses that predict majoring or minoring in computer science.

	Computer Science Major				Computer Science Minor			
Predictor	All Students	CS 1	CS ½	CS 0	All Students	CS 1	CS ½	CS 0
Pair programming	-.337*	-.309**	-.584	.334	-.123	.062	-.633	-.012
	(.143)	(.154)	(.362)	(.654)	(.308)	(.406)	(.641)	(.825)
Spring 2017 semester	-.197	-.138	-.212	-1.011	.314	-.420	.971	1.184
	(.199)	(.212)	(.500)	(1.126)	(.444)	(.598)	(.881)	(1.163)
Fall 2017 semester	-.024	.012	-.276	.125	.423	.193	.602	.488
	(.197)	(.213)	(.501)	(.718)	(.429)	(.539)	(.929)	(1.231)
Spring 2018 semester	-.296	-.337	-.357	-1.071	.146	-.002	.105	--
	(.196)	(.210)	(.487)	(1.126)	(.440)	(.545)	(1.012)	
CS ½ course	2.769***				1.058*			
	(.351)				(.538)			
CS 1 course	3.747***				1.604***			
	(.336)				(.468)			
Number of students	1,530	689	337	504	1,530	689	337	504
Number of discussion sections	96	41	26	29	96	41	26	29

Note. Standard errors are in parentheses. Outcomes were treated as binary in the analyses, and these were measured in Spring 2019 (or the last semester in which students were enrolled at the university). Semester 4 was omitted from the equation for predicting computer science minor among CS 0 students, since so few students who took that experimental course actually enrolled in the minor.
+*p* < .10 **p* < .05 ***p* < .01 ****p* < .001

Table 2. Results of multilevel analyses of pair programming predicting outcomes in subsequent computer science courses.

	Individual Computer Science Courses				Total Subsequent CS Courses	
Sample and Outcome	Discrete Structures	Data Structures	Informatics Programming	CS 1	0 versus 1 or More	# of Courses
All Students						
Course(s) Completed	-.274* (.134)	-.205 (.139)	.389 (.240)	-.201 (.249)	.047 (.057)	.115 (.138)
Course(s) Attempted	-.246+ (.133)	-.230 (.147)	.417+ (.229)	-.311 (.231)	.047 (.056)	.197 (.149)
Course Grade(s)	.102 (.102)	.113 (.100)	.269 (.235)	-.226 (.184)	.033 (.082)	
Only CS 1 Students						
Course(s) Completed	-.284+ (.156)	-.183 (.158)	--	--	.060 (.069)	.327+ (.175)
Course(s) Attempted	-.290+ (.156)	-.245 (.171)	--	--	.053 (.069)	.399* (.196)
Course Grade(s)	.073 (.112)	.039 (.107)	--	--	.063 (.165)	
Only CS ½ Students						
Course(s) Completed	--	--	.489+ (.289)	-.330 (.273)	.047 (.094)	-.178 (.287)
Course(s) Attempted	--	--	.530+ (.274)	-.475+ (.263)	-.016 (.087)	-.338 (.324)
Course Grade(s)	--	--	.278 (.258)	-.205 (.203)	.028 (.097)	

Note. Standard errors are in parentheses. The completed and attempted outcomes for individual courses were treated as binary, grades for individual courses were treated as continuous, and total number of courses completed and attempted were modeled via zero-inflated negative binomial regression analyses (which separately examine whether students took any coursework and then predict the number of courses among students who have taken at least one course). The analyses for the CS 1 course were limited to students who took either CS 0 or CS ½ for the experimental portion of the study, so that CS 1 constituted a subsequent course for these participants. The discrete structures and data structures courses are required for the computer science major, whereas the informatics course is required for the informatics major. All analyses controlled for semester; the analyses among all students also controlled for the experimental course.
+p < .10 *p < .05 **p < .01 ***p < .001

Table 3. Results of multilevel analyses for pair programming predicting outcomes within the introductory computer science course.

Outcome	All Students	CS 1	CS ½	CS 0
Course Grade	-.133*	-.091	-.257+	-.114
	(.057)	(.086)	(.139)	(.073)
Percent of Total Points Received	-.010	-.002	-.034+	-.008
	(.013)	(.016)	(.017)	(.009)
Percent on All Exams	-.012	-.008	-.030	-.002
	(.015)	(.018)	(.021)	(.011)
Percent on Final Exam	-.019	-.015	-.061*	.000
	(.018)	(.039)	(.027)	(.012)

Note. Standard errors are in parentheses. All analyses controlled for semester enrolled; the analyses among all students also controlled for the course taken during the treatment.
+*p* < .10 **p* < .05 ***p* < .01 ****p* < .001

Table 4. Results for multilevel analyses of pair programming predicting students outcomes among White students and Students of Color.

Outcome	White Students	Students of Color	Difference
Computer Science Major	-.505**	-.113	
	(.188)	(.205)	
Computer Science Minor	-.917*	.634	*
	(.459)	(.393)	
Completed Discrete Structures Course	-.533**	-.009	+
	(.178)	(.249)	
Completed Data Structures Course	-.643**	.185	**
	(.230)	(.200)	
Completed Informatics Programming Course	.296	.488	
	(.296)	(.451)	
Completed CS 1 (for CS 0 and CS ½ students)	-.486+	.501	+
	(.288)	(.442)	
Attempted Discrete Structures Course	-.396*	-.064	
	(.188)	(.231)	
Attempted Data Structures Course	-.575**	.100	*
	(.216)	(.220)	
Attempted Informatics Programming Course	.356	.462	
	(.281)	(.462)	
Attempted CS 1 (for CS 0 and CS ½ students)	-.565*	.298	+
	(.273)	(.396)	
Grade in Experimental Course	-.146*	-.119	
	(.067)	(.085)	

Note. Standard errors are in parentheses. All analyses controlled for semester and the experimental course. Differences in the effect of pair programming across subgroups were examined via an interaction term within analyses of all students. No significant main effects or interactions were observed for grades within subsequent courses or exam grades within the treatment course.
+$p < .10$ *$p < .05$ **$p < .01$ ***$p < .001$

Table 5. Mean differences in binary outcomes by pair programming experimental condition among White students.

	All Students				Students in CS 1			
Outcome	Individual	Paired	PP Diff	% Diff	Individual	Paired	PP Diff	% Diff
Computer Science Major	.275	.203	.072	35%	.523	.448	.075	17%
Computer Science Minor	.040	.017	.023	135%	.072	.029	.043	148%
Completed Discrete Structures Course	.251	.185	.066	36%	.542	.430	.112	26%
Attempted Discrete Structures Course	.258	.205	.053	26%	.549	.459	.090	20%
Completed Data Structures Course	.230	.160	.070	44%	.490	.378	.112	30%
Attempted Data Structures Course	.232	.166	.066	40%	.497	.384	.113	29%
	All Students				Students in CS ½			
Completed Informatics Course	.057	.071	-.014	-20%	.176	.226	-.050	-22%
Attempted Informatics Course	.064	.083	-.019	-23%	.204	.266	-.052	-23%
Completed CS 1	.138	.090	.048	53%	.315	.194	.121	62%
Attempted CS 1	.160	.100	.060	60%	.352	.202	.150	74%

Note. Unadjusted means are provided. PP Diff refers to percentage-point difference; % Diff refers to the percentage increase in the individual condition relative to the paired condition. The analyses for completing and attempting CS 1 among "all students" were restricted to students whose treatment course was CS 0 or CS ½, so that this outcome reflects subsequent coursework. The within-course analyses on the right-hand side of the table were conducted for the experimental course that had the highest mean value on that respective outcome.

Appendix. Descriptive statistics for all variables.

Variable	Mean	SD	Minimum	Maximum
Participated in pair programming	.52	.50	0	1
Treatment course was CS ½	.22	.41	0	1
Treatment course was CS 1	.45	.50	0	1
Treatment semester was Spring 2017	.22	.41	0	1
Treatment semester was Fall 2017	.24	.43	0	1
Treatment semester was Spring 2018	.24	.43	0	1
White/Caucasian	.60	.49	0	1
Female	.39	.49	0	1
Prior computer programming experience	.22	.26	0	1.33
Computer science major	.26	.44	0	1
Computer science minor	.04	.19	0	1
Completed Discrete Structures course	.26	.44	0	1
Completed Data Structures course	.23	.42	0	1
Completed Informatics Programming course	.06	.23	0	1
Completed CS 1 (among CS 0 and CS ½)	.12	.32	0	1
Attempted Discrete Structures course	.27	.45	0	1
Attempted Data Structures course	.24	.43	0	1
Attempted Informatics Programming course	.06	.24	0	1
Attempted CS 1 (among CS 0 and CS ½)	.13	.34	0	1
Grade in Discrete Structures course	2.63	1.02	0	4.33
Grade in Data Structures course	2.77	.91	0	4.33
Grade in Informatics Programming course	2.48	1.06	0	4.33
Grade in CS 1 (among CS 0 and CS ½)	2.45	1.05	0	4.33
Total Subsequent CS Courses Completed	1.33	2.20	0	14
Total Subsequent CS Courses Attempted	1.40	2.25	0	12
Average grade in subsequent CS courses	2.58	.97	0	4.33
Final letter grade in treatment course	2.75	.96	0	4.33
Proportion of points received in treatment course	.69	.20	.04	1.01
Proportion of points earned on all course exams	.65	.21	0	1.01
Proportion of points earned on final exam	.63	.24	0	1.01