Pair Programming in Perspective: Effects on Persistence, Achievement, and Equity in Computer Science

Nicholas A. Bowman , Lindsay Jarratt , K. C. Culver & Alberto M. Segre

JOURNAL OF RESEARCH ON EDUCATIONAL EFFECTIVENESS
https://doi.org/10.1080/19345747.2020.1799464

INTERVENTION, EVALUATION, AND POLICY STUDIES

Pair Programming in Perspective: Effects on Persistence, Achievement, and Equity in Computer Science

Nicholas A. Bowman[a], Lindsay Jarratt[a], K. C. Culver[b] and Alberto M. Segre[c]

[a]Department of Educational Policy and Leadership Studies, University of Iowa, Iowa City, Iowa, USA; [b]Pullias Center for Higher Education, University of Southern California, Los Angeles, California, USA; [c]Department of Computer Science, University of Iowa, Iowa City, Iowa, USA

ABSTRACT

Pair programming is a form of collaborative learning in computer science that involves two students working together on a coding project. Previous research has identified mostly positive outcomes from this practice, such as course grades and 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 achievement and persistence. However, the existing research findings are inconsistent, which may reflect limitations in research design and/or challenges with implementing pair programming in an ideal manner. The present study sought to provide rigorous evidence through a cluster-randomized trial with 1,530 undergraduates in 96 lab sections across three different introductory computer science courses. Within the full sample, pair programming was unrelated to virtually all outcomes. However, pair programming actually led to poorer outcomes among White students, including grades within the introductory course, attempting or completing subsequent computer science courses, and majoring or minoring in computer science. These negative effects were generally driven by White students whose partners had either low or high levels of prior programming experience.

ARTICLE HISTORY
Received 21 August 2019
Revised 1 July 2020
Accepted 13 July 2020

KEYWORDS
Pair programming; computer science; college students; undergraduate major; collaborative learning

Rapid advances in computer science and technology have had a broad and lasting impact across society. In an era of smartphones, 3 D 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 (e.g., Vee, 2013; Wing, 2008; Yadav et al., 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; Singer, 2019).

CONTACT Nicholas A. Bowman nick-bowman@uiowa.edu Department of Educational Policy and Leadership Studies, University of Iowa, N491 Lindquist Center, Iowa City, IA 52242, USA.

Supplemental material for this article can be accessed at https://doi.org/10.1080/19345747.2020.1799464.

Unfortunately, larger enrollments only exacerbate the challenges inherent in introductory computer science courses, especially for helping students acquire the programming skills needed 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 et al., 2008; Desjardins, 2015). These challenges are more present among underrepresented groups; similar to other STEM fields in the United States, computer science has struggled to increase and maintain the participation of women and Students of Color (Cheryan et al., 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 work collaboratively) has been offered as a means to increase student learning, grades, confidence, persistence, and satisfaction, but the causal impact of this pedagogical approach is unclear. This study brings rigorous evidence to bear on the effects of pair programming in the computer science classroom and its promise to improve student achievement and retention in computer science. It also explores the conditions and subgroups for which pair programming may be most (or least) effective at fostering desired outcomes.

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 et al., 2007; Faja, 2011; Hanks et al., 2011; Salleh et al., 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 et al., 2009; Kuppuswami & Vivekanandan, 2004; Zacharis, 2011) as well as cleaner code quality with fewer bugs (Begel & Nagappan, 2008; Bipp et al., 2008; Cliburn, 2003; Kuppuswami & Vivekanandan, 2004; Zacharis, 2011) and increased understanding of programming concepts (Begel & Nagappan, 2008; Faja, 2011; Howard, 2006). Others have found that pair programming is positively related to persistence in coursework (McDowell et al., 2006) as well as grades on tests, assignments, and the overall course (Chigona & Pollock, 2008; Kuppuswami & Vivekanandan, 2004; McDowell et al., 2006; Mendes et al., 2006; Wiebe et al., 2003). One group of researchers observed that students who were paired turned in their homework at significantly higher rates (Hanks et al., 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; Lai & Xin, 2011; McDowell et al., 2006; Nosek, 1998; VanDeGrift, 2004). Numerous studies suggest that individuals enjoy programming collaboratively more than working alone (Balijepally et al., 2009; Bipp et al., 2008; Cao & Xu, 2005; Cliburn, 2003; Faja, 2011; Hanks et al., 2004; Howard, 2006; McDowell et al., 2006; Nagappan et al., 2003; Nosek, 1998; Thomas et al., 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 et al. (2005) reported that programming professionals perceived pair programming to be less efficient than solo work in most instances, and several scholars have 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 that pair programming does not affect learning and understanding of concepts (Chigona & Pollock, 2008; Hanks et al., 2004), exam scores after controlling for prior achievement (Nagappan et al., 2003), or interest in computer science and plans to take subsequent computer science coursework (Bowman et al., 2019b).

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 et al., 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 in industry settings 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 et al., 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 et al., 2005; Cliburn, 2003; Nagappan et al., 2003; VanDeGrift, 2004; Wiebe et al., 2003); this incompatibility 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 et al., 2008; Sfetsos et al., 2009).

There are several possible explanations for the wide range of findings. First, some significant methodological problems have been present in this research. 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. At times, inferential statistical analyses were not used at all. Taken as a whole, these issues cast doubt on the validity of many previous findings. Second, one meta-analysis of the pair programming literature found signs of publication bias (Hannay et al., 2009), 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 and institutional contexts, were conducted for different lengths of time, were applied to different types of coding tasks, and included different types of participants. Finally, pair programming may be implemented in various ways that can notably affect the results (Coman et al., 2014).

Theoretical and Conceptual Framework for Equity in Computer Science

In U.S. higher education, women and Students of Color are consistently underrepresented in computer science (Estrada et al., 2016; Sax et al., 2017). Substantial research has attempted to explain and intervene in these dynamics, particularly the low enrollment of women. Potential precollege explanations for this disparity include women's stronger interpersonal orientation (Beyer, 2014; Ho et al., 2004; Sax et al., 2017), lower STEM confidence (Beyer et al., 2003; Sax et al., 2017) and lack of previous exposure to computing (Beyer, 2014; Cheryan et al., 2017; Sinclair & Kalvala, 2015). Explanations for the low representation of Students of Color in these majors are often similar, focusing on lack of prior content exposure or even the effects of affirmative action (see Harper, 2010, for a summary and critique of these narratives).

The present study focuses on outcomes that occur after students have opted into taking a computer science course. To do so, we draw from a view of college experiences and outcomes shaped by the Multicontextual Model for Diverse Learning Environments (Hurtado et al., 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 U.S. college campuses (Chang et al., 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 et al., 2017; Estrada et al., 2016). Additionally, scholars point to the ways in which STEM fields often reflect the language and norms of White, middle-class, and masculine discourse (Cheryan et al., 2009; Simon et al., 2017). Given this climate, it is perhaps unsurprising that women and Students of Color are less likely to identify with these fields (Hazari et al., 2013; Wong, 2015).

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 comprise only about 15% of 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-Juarez, 2016). While many of the explanations focus on gendered differences in prior experience, behavior, and interest (e.g., Beyer, 2014; Cheryan et al., 2017; Sinclair & Kalvala, 2015), other explanations highlight the pervasive masculine culture and hostile learning environments in many 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 research 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 (Prinsen et al., 2007).

Likewise, Students of Color—specifically Black, Latinx, and Indigenous students—remain underrepresented in computer science programs (Malcom & Feder, 2016; also see Museus et al., 2011). 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 et al., 2009). Some research has also linked a field's pervasive beliefs about innate brilliance and ability with levels of underrepresentation (Leslie et al., 2015), perhaps because this belief leads to classroom practices that are 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).

Asian American and international students are often well-represented numerically in STEM courses and majors (and therefore underrepresented in STEM equity literature), but they may face a distinct yet related set of challenges. For instance, Asian and Asian American students may experience pressure via expectations to serve as model students (e.g., McGee et al., 2017). Additionally, an aggregate picture that highlights their relative success in the STEM classroom can obscure notable variation within this group (Museus et al., 2011). International students must also navigate considerable differences in culture (Pedersen et al., 2016), feelings of loneliness (Cho & Yu, 2015), challenges communicating in a second language (Luo et al., 2019), and making connections with U.S. peers (Pedersen et al., 2016). Furthermore, international students who are racial minorities may face discrimination and xenophobia due to their perceived race and nationality (e.g., Yao et al., 2019).

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 large effects in smaller class sizes (Freeman et al., 2014). The best available evidence 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). For instance, the incorporation of an interactive workshop within an introductory biology course 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 with PowerPoint lectures within a veterinary course; similarly, female students benefited more from the team-based approach than did male students (Malone & Spieth, 2012). A study of pair programming found that international students benefited from being paired with U.S. domestic students in terms of completing a larger portion of the paired assignment and feeling more confident about the quality of their code, whereas domestic students did not exhibit benefits from cross-national pairing (Bowman et al., 2020).

A fair portion of the research over the last two decades indicates that collaborative paired work as an active learning strategy is particularly effective in computer science classrooms (Braught et al., 2011). In the case of pair programming, this collaborative approach to learning may counter isolation and competition in the classroom, especially for students who are marginalized in the learning space. Engaging in meaningful interactions across difference is a highly effective approach for improving intergroup attitudes within and outside of the classroom (Bowman, 2011; Pettigrew & Tropp, 2006), 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 student outcomes (Coman et al., 2014; Wiebe et al., 2003; Williams & Kessler, 2002). For instance, one study observed less clear role differentiation than that proposed by proponents of pair programming (Bryant et al., 2008). Other research indicates that dominating behavior is more likely when there is a mismatch in gender (Williams et al., 2002) or in achievement and skill (Chaparro et al., 2005; Hanks et al., 2011; Stephens & Rosenberg, 2003; Williams et al., 2002), which 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 constraints can all lead to disengagement in pairs (Plonka et al., 2012).

Present Study

This study sought to address the challenges of previous research through a rigorous examination of the impact of pair programming. We conducted a cluster-randomized trial that assigned lab sections to either pair programming or traditional programming; this experimental design leads to much stronger causal inferences than in previous work. We also implemented pair programming over four semesters in 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 undergraduates and nearly 100 lab sections, so the analyses had sufficient statistical power to detect main effects among all students and separately by course. 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 to 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. In doing so, we also explored whether grades in the initial computer science course or partners' prior experience with computer programming may help explain any subsequent effects.

Method

Study Context, Procedure, and Participants

This study included undergraduates who took an introductory computer science course 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 $^{1}/_{2}$). These courses varied 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 $^{1}/_{2}$) 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 $^{1}/_{2}$) per semester.

Lab sections were randomly assigned to engage in pair programming or in individual programming practices. Aside from this pedagogical practice, 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. The researchers and primary course instructors trained the TAs on how to facilitate pair programming. Students in the experimental condition switched partners about every five weeks so that they participated in three different pairings throughout the semester; this frequency was intended to give students enough time to become comfortable with their partner and 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 meeting outside of class prevented this approach from being viable). Students in paired sections were randomly assigned to partners, which were reassigned twice during the semester (giving each student a total of three pairings over the course of the semester). As discussed later, our choice to assign randomly may have affected the overall effectiveness of the intervention, but random assignment also facilitated stronger causal claims from the results. Guidelines for effective pair programming have recommended periodic reassignment of pairs (Williams et al., 2008), and this practice also minimizes the chance of being assigned an intractable or incompatible partner for the entirety of the semester.

Students in paired sections were instructed to use pair programming in a method consistent with formal definitions of this practice: Both students would sit at the same computer and share the same keyboard. 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. TAs were trained to monitor and encourage effective pair work, and students were also given a handout with tips and strategies for working in this manner.

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 the 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 (a few 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 attended postsecondary education), 60% were White/Caucasian, 20% were international students (the vast majority 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 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, and high school GPA, $ps > .10$ (see the supplemental material in Table S1).

Measures

Two of the primary outcome variables indicated whether students had a major or a minor in computer science in the most recent semester available. 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.

Other outcomes examined students' subsequent computer science coursework. We obtained data on all computer science courses that students completed (with a passing grade) or attempted (with any grade, including F or W). This information was used to compute variables for individual courses as well as the total number of computer science courses attempted and completed. Several binary variables indicated students' enrollment in several key courses: Discrete Structures and Data Structures (both required for computer science majors and are prerequisites for taking various other courses in the major); Programming for Informatics (required for informatics majors, but not computer science majors); and CS 1 (this outcome was only examined among students enrolled in CS 0 or CS $^1/_2$ during their treatment semester). Three variables were created for each course: attempting the course (0 = no, 1 = yes), successful course completion (0 = no, 1 = yes), and the grade that students earned (F = 0 to A+ = 4.33). The grade within the treatment introductory course was examined with the same GPA scale.

The primary independent variable was whether students participated in a lab section that used pair programming (0 = no, 1 = yes). Dummy variables indicated the course in which they were enrolled during the study (CS $^1/_2$ and CS 1, with CS 0 as the referent group) and semester in which they participated (Spring 2017, Fall 2017, and Spring 2018, with Fall 2016 as the referent group). The primary moderators were students' race/ethnicity (0 = Student of Color, 1 = White/Caucasian) and sex (0 = male, 1 = female). Supplemental analyses used dummy variables for subgroups of Students of Color: underrepresented racial minority students (i.e., American Indian/Alaska Native, Black/African American, Latinx/Hispanic, and Pacific Islander/Native Hawaiian students), international students, and students of primarily Asian descent (Asian American and international students, the vast majority of whom were from Asian countries).

For additional analyses, the treatment condition was divided into three separate groups based on partners' prior experience with computer programming before the course. In the first two weeks of the semester, students completed a survey that contained nine items about prior experience with website design; programming mobile apps; and using BASIC, C/C++, Java, Javascript, Perl, Python, and Ruby. All items used a four-point scale (1 = none, to 4 = a lot), and the Cronbach's alpha for this index was .76. Given the right skew of the resulting distribution, the natural log of the original measure was used; the average of participants' three partners was computed. Descriptive statistics for all variables appear as supplemental material (Table S2).

Analyses

Hierarchical linear modeling (HLM) analyses were conducted to account for the nesting of students (level 1) within lab sections (level 2) within teaching assistants (level 3; see Raudenbush & Bryk, 2002; Snijders & Bosker, 2012). Students 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. Preliminary analyses showed that the results reported here were substantively identical when making alternative choices to account for the multilevel structure of the data (e.g., ignoring the clustering within TAs, conducting regression analyses with 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), so these outcomes were 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). Grades within the introductory course and subsequent courses were treated as continuous via traditional hierarchical linear modeling analyses.

Preliminary analyses examined interactions between pair programming and each of the key demographic characteristics (race or sex). Significant interactions in the same direction were frequently identified for race, so subgroup analyses by race were conducted (interactions by sex were rarely significant). Additional subgroup analyses were also conducted to explore the impact of pair programming within each course. Two sets of follow-up analyses were conducted within the subgroup that exhibited various significant effects of pair programming (i.e., White students) to better understand the processes and conditions that may have contributed to these findings. (Note that the patterns of results for all students are somewhat similar, except that significant effects are less prevalent within this full participant sample.) First, analyses considered the potential role of grades in the introductory course as a mediator of significant relationships between pair programming and subsequent outcomes. Bias-corrected bootstrap analyses with 5,000 resamples examined the indirect effects of pair programming on longer-term outcomes via introductory course grades (see Hayes, 2018). Second, additional multilevel analyses of White students separated the treatment condition into three groups: students whose partners had low prior experience with computer programming (on average), those with medium experience, and those with high experience. The cutoff points were chosen so that these three treatment groups had roughly equal sample sizes. Preliminary analyses found that the results for partners' prior experience generally did not interact with participants' own experience (i.e., the correspondence or matching of students' and their partners' experience did not provide additional information). The moderation, subgroup, and mediation analyses all controlled for semester and course.

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) over four semesters and by exploring potential moderation effects across student and partner characteristics, but the results may not apply to other

Table 1. Unstandardized coefficients for multilevel analyses predicting grades in the introductory computer science course.

Predictor	All Students	CS 1	CS $^1/_2$	CS 0
Pair programming	−.133* (.057)	−.091 (.086)	−.257+ (.139)	−.114 (.073)
Spring 2017 semester	.035 (.080)	.090 (.118)	.069 (.196)	−.076 (.104)
Fall 2017 semester	.002 (.078)	−.064 (.121)	−.220 (.194)	.180 (.096)
Spring 2018 semester	.001 (.078)	−.057 (.118)	.059 (.185)	.030 (.102)
CS $^1/_2$ course	−.833*** (.077)			
CS 1 course	−.478*** (.065)			
Number of students	1,308	537	278	493
Number of discussion sections	96	41	26	29

Note. Standard errors are in parentheses. Students who withdrew from the course did not receive an A–F letter grade (for which grade points are assigned) and therefore were not included in these analyses.
+$p < .10$; *$p < .05$; **$p < .01$; ***$p < .001$.

institutions or to other implementations of pair programming. Later in the paper, we discuss various aspects of pair programming implementation, including features of this university's approach to pair programming that we believe are most likely to have affected the findings. Furthermore, although we were able to examine various post-course outcomes, a longer time period would have allowed us to consider graduation data for all students.

Results

What Are the Main Effects of Pair Programming Overall and by Introductory Course?

Table 1 contains results for multilevel analyses that examined the impact of pair programming on grades within the introductory course. In the full sample, students who participated in lab sections with pair programming had significantly lower grades than did students in lab sections with individual programming. When examining grades separately by introductory course, pair programming had a marginally significant negative effect on course grades in CS $^1/_2$ ($p < .10$), but the results for CS 0 and CS 1 were not significant.

The results for pair programming predicting participation and success in computer science coursework after the treatment semester are shown in Table 2. 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 $^1/_2$ and Programming for Informatics, so subgroup analyses were conducted predicting this outcome for those who took CS $^1/_2$). Only two of the 39 analyses were significant at a threshold of $p < .05$, which is consistent with what one might expect to find via random chance. Both coefficients indicated adverse effects of the treatment: Pair programming was negatively associated with completing the Discrete Structures course among all students, and it predicted a greater likelihood of not attempting any subsequent computer science coursework among students who initially took CS 1. Some additional patterns emerged when using a more lenient threshold for statistical significance ($p < .10$): pair programming predicted a lower likelihood of completing or attempting Discrete Structures among all students and those who started in CS 1, whereas pair programming was positively related to completing and attempting the Programming for Informatics

Table 2. Unstandardized coefficients for multilevel analyses examining the effect of pair programming on 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)	−.203 (.139)	.390 (.240)	−.201 (.249)	.115 (.138)	.047 (.057)
Course(s) attempted	−.246+ (.133)	−.230 (.147)	.402+ (.238)	−.311 (.231)	.197 (.149)	.047 (.056)
Course grade(s)	.102 (.102)	.113 (.100)	.269 (.235)	−.226 (.184)	.033 (.082)	
Only CS 1 students						
Course(s) completed	−.284+ (.156)	−.183 (.158)	–	–	.327+ (.175)	.060 (.069)
Course(s) attempted	−.290+ (.156)	−.245 (.171)	–	–	.399* (.196)	.053 (.069)
Course grade(s)	.073 (.112)	.039 (.107)	–	–	.028 (.097)	
Only CS $^1/_2$ students						
Course(s) completed	–	–	.489+ (.289)	−.330 (.273)	−.178 (.287)	.047 (.094)
Course(s) attempted	–	–	.523+ (.288)	−.474+ (.263)	−.338 (.324)	−.016 (.087)
Course grade(s)	–	–	.278 (.258)	−.206 (.203)	.063 (.165)	

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 those who have taken at least one course). For the portion of the zero-inflated analyses examining any subsequent coursework, positive values mean that pair programming is associated with not taking future computer science courses. The analyses for the CS 1 course were limited to students who took either CS 0 or CS $^1/_2$ as their introductory course, so 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 Programming for Informatics course is required for the informatics major. All analyses controlled for semester; the analyses among all students also controlled for the introductory computer science course.
+$p < .10$; *$p < .05$; **$p < .01$; ***$p < .001$.

course among students who originally took CS $^1/_2$ as well as attempting programming for Informatics among all students. At this same threshold of significance, pair programming was also associated with not completing any additional computer science courses among CS 1 students, and it was negatively related to attempting CS 1 among CS $^1/_2$ students.

Table 3 provides the findings for the effects of pair programming on having a computer science major or minor. In the full sample, students in pair programming were significantly less likely to be computer science majors than those who participated in individual programming; this difference across experimental conditions was 5.2% points. When examining each course separately, this same significant pattern was observed for students who originally enrolled in CS 1: students in pair programming were 7.6% points less likely to be computer science majors than were those in individual programming. No other effects were significant.

In summary, pair programming resulted in lower grades within the introductory course and a reduced likelihood of majoring in computer science. Pair programming was unrelated to minoring in computer science, and it had sparse relationships with completing and taking future computer science coursework, especially when using a statistical significance criterion of $p < .05$. Patterns of marginally significant results suggest that pair programming reduced the likelihood of attempting or completing the Discrete Structures class, whereas it improved the chances of taking the programming for Informatics class, especially among students in CS $^1/_2$.

Table 3. Unstandardized coefficients for multilevel analyses that predict majoring or minoring in computer science.

	Computer science major				Computer science minor			
Predictor	All students	CS 1	CS $^1/_2$	CS 0	All students	CS 1	CS $^1/_2$	CS 0
Pair programming	−.337*	−.309**	−.580	.334 (.654)	−.122	.060 (.379)	−.633	−.012
	(.143)	(.154)	(.361)		(.307)		(.641)	(.825)
Spring 2017 semester	−.197	−.138	−.212	−1.010	.315	−.418	.971	1.184
	(.199)	(.212)	(.500)	(1.126)	(.444)	(.610)	(.881)	(1.163)
Fall 2017 semester	−.025	.012	−.274	.125	.423	.199	.602	.488
	(.197)	(.213)	(.501)	(.718)	(.431)	(.579)	(.929)	(1.231)
Spring 2018 semester	−.296	−.227	−.357	−1.071	.139	−.086	.105	–
	(.196)	(.210)	(.487)	(1.126)	(.456)	(.601)	(1.012)	
CS $^1/_2$ course	2.769***				1.055			
	(.352)				(.546)			
CS 1 course	3.747***				1.602***			
	(.336)				(.471)			
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 introductory course actually enrolled in the minor.
$^+p < .10$; $^*p < .05$; $^{**}p < .01$; $^{***}p < .001$.

Table 4. Unstandardized coefficients for multilevel analyses examining the effect of pair programming on outcomes among White students and Students of Color.

Outcome	White students	Students of Color
Grade in introductory CS course	−.146* (.067)	−.119 (.085)
Completed discrete structures course	−.533** (.198)	−.009 (.209)
Completed data structures course	−.643** (.230)	.185 (.200)
Completed informatics programming course	.296 (.296)	.322 (.459)
Completed CS 1 (for CS 0 and CS $^1/_2$ students)	−.486+ (.288)	.501 (.442)
Attempted discrete structures course	−.396* (.188)	−.064 (.231)
Attempted data structures course	−.575** (.216)	.100 (.220)
Attempted informatics programming course	.356 (.281)	.298 (.441)
Attempted CS 1 (for CS 0 and CS $^1/_2$ students)	−.565* (.273)	.309 (.458)
Computer science major	−.505** (.188)	−.113 (.205)
Computer science minor	−.917* (.459)	.627 (.399)

Note. Standard errors are in parentheses. All analyses controlled for semester and the introductory course. No significant main effects were observed for grades within subsequent courses.
$^+p < .10$; $^*p < .05$; $^{**}p < .01$; $^{***}p < .001$.

Who Does Pair Programming Harm or Help?

The impact of pair programming clearly differs as a function of students' race. As shown in Table 4, pair programming contributed to various negative outcomes among White students, including (a) a lower grade within the introductory computer science course; (b) a lower likelihood of completing or attempting Discrete Structures, Data Structures, and CS 1 (the analyses for CS 1 only used students who initially took CS 0 or $^1/_2$); and (c) a lower likelihood of majoring or minoring in computer science. Conversely, pair programming was not significantly related to any outcome among Students of Color, and no significant effects were observed for grades in subsequent courses within either racial subgroup. Additional analyses (not shown here) also found virtually no significant effects of pair programming among underrepresented racial

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

	All White students				White students in CS 1			
Outcome	Individual	Paired	PP diff	96 diff	Individual	Paired	PP diff	96 diff
Computer science major	.275	.203	.072	3596	.523	.448	.075	1796
Computer science minor	.040	.017	.023	13596	.072	.029	.043	14896
Completed discrete structures course	.251	.185	.066	3696	.542	.430	.112	2696
Attempted discrete structures course	.258	.205	.053	2696	.549	.459	.090	2096
Completed data structures course	.230	.160	.070	4496	.490	.378	.112	3096
Attempted data structures course	.232	.166	.066	4096	.497	.384	.113	2996
	All White students				White students in CS ½			
Completed informatics course	.057	.071	−.014	−2096	.176	.226	−.050	−2296
Attempted informatics course	.064	.083	−.019	−2396	.204	.266	−.052	−2396
Completed CS 1	.138	.090	.048	5396	.315	.194	.121	6296
Attempted CS 1	.160	.100	.060	6096	.352	.202	.150	7496

Note. Unadjusted means are provided. PP Diff refers to percentage-point difference; 96 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 White 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.

minority students, international students, and students of primarily Asian descent; the lone exception was a marginally significant positive result ($p < .10$) for completing Programming for Informatics among international students. These divergent results were not simply the function of disparate sample sizes, as Students of Color as a whole comprised 40% of all participants.

Because the vast majority of the outcomes presented in Table 4 were 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 White students in all courses, many of these differences fell near effect size guidelines for college impact studies that indicate small relationships (5% points) or were approaching a medium effect size (9% points; see Mayhew et al., 2016). For example, White students in the individual programming condition were 7.2% points more likely to major in computer science than those in the pair programming condition. These percentage-point effect sizes were 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 increased from 6.6% points among all White students to 11.2% points among White students who took CS 1 during the treatment semester. The largest effect size occurred for attempting CS 1 among students who originally took CS ½; this 15%-point increase is equal to Mayhew et al.'s guideline for a large effect. The lone exception to these patterns is that pair programming led 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%-point increase in White students' having a minor in computer science, which might seem like a trivial amount. However, given the small percentage of

Table 6. Unstandardized coefficients for bootstrap mediation analyses examining the indirect effect of pair programming on student outcomes via introductory course grades among White students.

	Indirect effect of pair programming		Direct effects	
Outcome	8 (SE)	9596 confidence interval	Pair programming **	Course grades ***
Completed discrete structures course	−.102 (.048)	[−.211, −.021]	−.579** (.218)	.705*** (.130)
Completed data structures course	−.091 (.045)	[−.191, −.012]	−.699 (.224)	.627*** (.131)
Completed CS 1 (for CS 0 and CS $^1/_2$ students)	−.159 (.086)	[−.346, −.010]	−.274* (.316)	.978*** (.207)
Attempted discrete structures course	−.095 (.047)	[−.200, −.017]	−.497** (.212)	.656*** (.126)
Attempted data structures course	−.085 (.041)	[−.182, −.017]	−.641 (.220)	.583*** (.128)
Attempted CS 1 (for CS 0 and CS $^1/_2$ students)	−.152 (.082)	[−.335, −.012]	−.392** (.297)	.930*** (.195)
Computer science major	−.071 (.037)	[−.160, −.014]	−.575+ (.201)	.486** (.115)
Computer science minor	−.125 (.069)	[−.302, −.025]	−.829 (.448)	.844 (.292)

Note. Significant indirect effects for bootstrap analyses were determined through 9596 confidence intervals that do not include zero; therefore, all indirect effects displayed in this table were significant. The analyses controlled for semester and the introductory course when predicting both the mediator and the post-course outcome. Because these analyses sought to explain the adverse effects of pair programming, only outcomes that exhibited significant effects of pair programming among White students were examined.
+p < .10; *p < .05; **p < .01; ***p < .001.

students who enrolled in that minor, White students in the individual condition were 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 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 occurred for attempting CS 1 among White students who started in CS 0 or CS $^1/_2$ (60%) and solely those who started in CS $^1/_2$ (74%).

In summary, the impact of pair programming varied systematically by race, such that it reduced grades in the introductory course, completing and attempting subsequent computer science courses, and majoring and minoring in computer science among White students. These relationships were all non-significant among Students of Color. The negative effects among White students were sometimes substantial in size when examining logical course sequences.

What Explains the Negative Effects of Pair Programming among White Students?

Two sets of supplemental analyses were conducted in an attempt to shed light on the prevalent significant results among White students. First, grades within the introductory computer science course were used as a mediator to explain the lower outcomes in subsequent courses. As shown in Table 6, these bootstrap mediation analyses identified significant indirect effects of pair programming via grades on all eight distal outcomes for which direct effects were initially observed. However, a mediator is primarily informative if it explains a substantial portion of the direct effect (Kenny, 2018), but that was generally not the case for introductory course grades. The direct effect of pair programming remained significant in six out of the eight outcomes in the mediation analyses, and these coefficients for pair programming were similar in size to those that did not include the mediator. Two direct effects of pair programming became nonsignificant when adding grades to the model for completing and attempting CS 1 among White

Table 7. Unstandardized coefficients for multilevel analyses of the effects of pair programming by level of partners' average prior programming experience among White students.

	Comparisons of pair programming versus individual programming		
Outcome	Low partner experience	Medium partner experience	High partner experience
Grade in introductory CS course	−.210* (.087)	.021 (.087)	−.193* (.098)
Completed discrete structures course	−.896** (.323)	−.170 (.266)	−.615* (.277)
Completed data structures course	−.844* (.356)	−.123 (.298)	−.950** (.324)
Completed informatics programming course	−.287 (.463)	.473 (.380)	.705+ (.410)
Completed CS 1 (for CS 0 and CS $^1/_2$ students)	−.751+ (.431)	−.411 (.375)	−.379 (.402)
Attempted discrete structures course	−.788* (.310)	−.025 (.258)	−.439+ (.263)
Attempted data structures course	−.848* (.344)	−.021 (.283)	−.858** (.308)
Attempted informatics programming course	−.279 (.440)	.667+ (.354)	.668+ (.398)
Attempted CS 1 (for CS 0 and CS $^1/_2$ students)	−.988* (.427)	−.546 (.361)	−.617 (.393)
Grade in discrete structures course	.297 (.254)	.334+ (.189)	−.070 (.199)
Grade in data structures course	.114 (.225)	.061 (.168)	.148 (.191)
Grade in informatics programming course	.559 (.472)	.150 (.351)	−.058 (.385)
Grade in CS 1 (for CS 0 and CS $^1/_2$ students)	.559 (.387)	.217 (.332)	−.264 (.358)
Computer science major	−1.007** (.316)	−.069 (.255)	−.626* (.272)
Computer science minor	−.971 (.776)	−.751 (.655)	−.852 (.666)

Note. Standard errors are in parentheses. The three treatment variables for pair programming with partners who had low, medium, or high prior experience with computer programming were entered simultaneously in multilevel analyses; the control condition (individual programming) served as the referent group. All models controlled for semester and the introductory course.
$^+p < .10$; $^*p < .05$; $^{**}p < .01$; $^{***}p < .001$.

students who initially took CS $^1/_2$ or CS 0. However, these coefficients for the treatment were reduced by less than 50% with the inclusion of grades, which suggests that introductory course grades may not completely explain the effect of pair programming.

A second set of analyses divided the treatment condition into three groups based on partners' prior programming experience, since partners were also randomly assigned among students within pair programming. As shown in Table 7, White students whose partners had either low *or* high prior experience (on average) exhibited various negative outcomes relative to engaging in individual programming. Specifically, students with these extremes of partner experience had lower grades in the introductory course, were less likely to have completed or attempted the Data Structures course, were less likely to have completed the Discrete Structures course, and were less likely to major in computer science ($ps < .05$). Moreover, students with lower-experience partners were also less likely to attempt Discrete Structures or CS 1. Relaxing the statistical significance criterion to $p < .10$ led to additional findings. Some patterns were consistent with those results: programming with high-experience partners reduced the likelihood of attempting Discrete Structures, and having low-experience partners predicted less completion of CS 1 (among students who started in CS 0 and CS $^1/_2$). However, other marginally significant findings were favorable for pair programming (i.e., high-experience partners and completing programming for Informatics, medium- and high-experience partners for attempting that same course, and medium-experience partners and grades in Discrete Structures). Supplemental analyses among Students of Color found no significant results of pair programming by partners' prior experience level at $p < .05$, and only one result was significant at $p < .10$.

In summary, pair programming has an indirect effect via introductory course grades on various post-course outcomes among White students, but this short-term mediator does not explain the majority of the main effects on those longer-term outcomes. When dividing the treatment into three groups based on partners' prior experience with computer programming, students who engaged with partners who had either low or high levels (on average) of prior experience had numerous negative outcomes relative to students who programed individually, whereas students with partners who had a medium level of experience did not exhibit any adverse outcomes relative to individual programming.

Discussion

This cluster-randomized trial found that pair programming has virtually no short-term or long-term benefits for students' grades or their subsequent engagement with computer science coursework and degree programs. If anything, pair programming led to a modest reduction in grades for the introductory course and the likelihood of majoring in computer science among all students. Moreover, the findings were frequently negative among White students, as pair programming resulted in non-trivial reductions in course grades as well as future participation in computer science courses, minor, and major. These adverse results were driven by partners who had either low or high average levels of prior experience with computer programming.

Divergence in Findings and Methodology Compared to Previous Research

This study expands and improves upon earlier studies in several ways. First, the random assignment of lab sections to experimental conditions led to much stronger conclusions about the efficacy of pair programming. Second, in a related issue, students in the paired lab sections were randomly assigned to partners, which allowed us to explore the impact of partners' prior programming experience. Third, this paper identified notable differences in the impact of pair programming by race, whereas this demographic attribute has received surprisingly little attention in pair programming research. Fourth, this study collected data over an extended timeframe, so various outcomes were observed up to 2 $\frac{1}{2}$ years after the introductory course, whereas previous studies have often focused on within-course outcomes. Finally, the examination of a large number of students and lab sections provided sufficient statistical power to detect even small effects (if they were present) and to conduct meaningful subgroup analyses.

The present results contrast with research that generally obtains positive results for pair programming within both educational and workplace settings (although substantial heterogeneity does exist; see Dyba et al., 2007; Faja, 2011; Hanks et al., 2011; Salleh et al., 2011; Umapathy & Ritzhaupt, 2017). Specifically, these findings diverge from the generally positive results for studies on pair programming and students' grades (Chigona & Pollock, 2008; Kuppuswami & Vivekanandan, 2004; McDowell et al., 2006; Mendes et al., 2006; Umapathy & Ritzhaupt, 2017; Wiebe et al., 2003); persistence in computer science coursework (McDowell et al., 2006); or decision to major in computer science (Hanks et al., 2011; McDowell et al., 2006; Nagappan et al., 2003; Umapathy & Ritzhaupt, 2017).

Why do the present findings differ from prior research? Several factors may have contributed individually or in combination. Hannay et al. (2009) meta-analysis found that publication bias was a notable concern, so additional papers with nonsignificant or negative findings for pair programming likely exist, but these may not be publicly available. Additionally, methodological issues with prior research undermine the credibility of many positive findings. A surprising number of previous 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 previous published research may overstate the strength of the association. In a sense, selection bias may have been a notable problem not only for the assignment of students to treatment within studies, but also for the public presentation of research findings through peer-reviewed publications.

That said, we are aware of few studies that utilized a similar research design to our own; the most closely aligned research was presented in Wiebe et al. (2003) and Williams et al. (2003). Similar to our current work, their family of experiments utilized a cluster-randomized trial over four semesters, had students work in a total of three pairings, obtained large sample sizes, and utilized appropriate control measures; these features make that research some of the strongest to date. Nonetheless, many of our results are inconsistent with their generally positive findings. This divergence may be at least partially attributable to differences in the definition of outcome measures (e.g., examining "passing" grades as a binary outcome, as opposed to grades as a continuous outcome) or to differences in the implementation of pairing (e.g., allowing student input and attempting to match compatible programmers versus random assignment).

In a potentially favorable pattern for pair programming in the present study, several analyses did yield positive results for the link between this treatment and completing or attempting the Programming for Informatics course. These relationships were marginally significant ($.05 < ps < .10$), so we urge caution in this interpretation. As noted earlier, this course is a requirement for informatics and other data science majors, but not for the computer science major. The vast majority of students who enroll in this subsequent course took CS $^1/_2$ in the treatment semester, since that is a prerequisite for programming for Informatics. The positive results may be especially surprising, then, since pair programming had a negative effect on grades for CS $^1/_2$, but not for the other two introductory courses. It is possible that many students in CS $^1/_2$ are debating whether to enter computer science majors (a decent number of these students enroll in CS 1 afterwards), so the positive effects for the informatics course may actually reflect the fact that pair programming diverted some students away from majoring in computer science. Consistent with this possibility, pair programming had a marginally significant negative effect on attempting CS 1 among students who originally took CS $^1/_2$; this inverse relationship was fairly large among White students.

Explaining the Negative Results for Pair Programming

When considering the reasons for the negative effects observed here, it is possible that the implementation of pair programming in the present study was 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 the 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, perhaps undermining the necessary coordination that collaborative learning should require (Barron, 2000). In a 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 a built-in timeline for leaving a partner who they might not like), and this approach is recommended as a best practice for pair programming (Williams et al., 2008). However, these changes may have made it more difficult for students to adjust to working with their partner effectively. That said, surveys at the end of the semester found that the subjective experiences of students and TAs within these pair programming sections were mostly positive (consistent with most previous research), suggesting that pair programming may have been implemented in a reasonable manner.

Beyond the specifics of this study, pair programming may provide more opportunities for problems that can undermine its effectiveness relative to other forms of collaborative learning. Many collaborative learning activities are short-term in nature, involve several students, and do not require every student to contribute meaningfully. Thus, it may not be a substantial problem in many circumstances if one group member (out of 4–5 total) does not show up for class, has not prepared or completed work in advance, and/or does not engage substantially or productively in group conversations. In contrast, these problems within pair programming would make this practice equivalent to independent programming at best and would hinder the learning and interest of both students at worst. Indeed, in their meta-analysis of collaborative learning activities, Tomcho and Foels (2012) found that long-term duration of group work and completing group (rather than individual) graded assignments, which are generally characteristic of pair programming, were associated with reduced learning and achievement. Group-level accountability can lead to one student doing much of the work, which may have adverse effects for that student (who could become frustrated or overwhelmed) as well as the other student(s) (who may not learn or develop an interest in or mastery of the material).

The fact that pair programming exhibited various negative outcomes among White students, but not among Students of Color, is quite notable. Numerous negative results were obtained for grades within the introductory course, for participation in future coursework, and for enrolling in computer science degree programs. However, in another paper that employed this same dataset, we actually found no moderation by race on students' computer science interest, perceptions, intentions, and completion of these introductory courses. The present paper also observed no effects on student grades in subsequent courses. Taken together, these patterns suggest that pair programming primarily affects White students' longer-term participation in computer science coursework rather than their outcomes within the courses that they do take.

The reasons for this racialized pattern are not entirely clear. 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 (e.g., Malone & Spieth, 2012; Preszler, 2009); the present results exhibited this same relative difference across groups, but without the presence of positive effects for any group. The negative findings for White students may stem, at least in part, from the fact that computer science classrooms were more racially diverse than the undergraduate population of the university and the surrounding community, especially in terms of the number of Asian international 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 et al., 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 model students (e.g., McGee et al., 2017). If White students perceive this stereotype, then interacting with Students of Color via pair programming may have heightened the threat that White students could experience within this coursework. Preliminary 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 challenge the presumption of superiority that White students may hold in academic contexts (see Walton & Cohen, 2003).

Although significant indirect effects via introductory course grades were observed, this mediator did not substantially reduce the effects for pair programming when predicting most outcomes. Grades in the introductory course seemed to constitute a plausible mediator for multiple reasons. Many students perceive grades as providing meaningful feedback about their competence and abilities; students then use this information to decide whether to persist in their major or in college at all (Stinebrickner & Stinebrickner, 2012). As a result, the lower course grades may lead students to change their mind about pursuing a computer science major. Furthermore, even if students want to continue within computer science, those who receive low grades may be prevented from taking future coursework as a result of academic probation and/or dismissal from the major or the university.

Regarding other potential mediators, we have attempted to identify psychological effects of pair programming in prior analyses of these data using a student survey administered at the end of the introductory course (Bowman et al., 2019b). We found that pair programming had no impact on students' interest in computer science, computer programming, or course content; their comfort with, confidence about, or anxiety toward computer science; or their plans to take subsequent computer science courses. Moreover, none of these relationships was significantly moderated by race/ethnicity. Additional analyses among White students showed that pair programming was not significantly related to any of those survey-based short-term outcomes, so these constructs cannot mediate the negative effects that were observed here.

Perhaps the most promising insights for explaining the negative effects come from analyses that separated the pair programming treatment into three levels based on partners who had low, medium, and high average levels of prior experience with computer programming. Interestingly, negative results corresponded to having partners with either low or high levels of experience, whereas no negative results occurred for having partners with a medium amount of experience. This set of findings may be consistent with the meta-analysis conducted by Dyba et al. (2007), which concluded that the interaction of programmer expertise and task complexity may dictate whether pair programming is an appropriate method. Indeed, experienced partners may take too much of a lead role, thereby depriving the other student of the opportunity to engage with and learn relevant programming skills and course content (Bowman et al., 2019a; Stephens & Rosenberg, 2003). Programmers with greater skill are also more likely to become frustrated with challenges that arise in pair programming (Chaparro et al., 2005), which may lead to poorer collaboration among pairs. On the other hand, novice programmers talk much more frequently in pairs than do experienced programmers (Bryant, 2004), which may be especially problematic when the assignments must be completed during a 50-minute lab section. Some of these novice programmers may not have sufficient knowledge to engage meaningfully in pair programming. In these situations, the partner may spend considerable time teaching (if they have sufficient expertise), or they may be unable to provide meaningful assistance (if they also have little experience), both of which would lead to far less enjoyable and productive engagement.

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 frequently 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). The current findings about partners' prior experience provide further evidence about these interpersonal dynamics based on randomly assigned pairings.

The extent to which the present study may generalize to other contexts is an important issue. Although pair programming has some key features that are central to its design, Williams et al. (2008) offer 11 guidelines for effective implementation of pair programming, which implies that there are numerous decisions that could potentially alter the effectiveness of this practice. These guidelines do not even constitute a complete list of possible choices. For instance, one approach could be to have students engage in pair programming earlier in the semester or earlier within a portion of the course and then move to individual programming in lab sections as their skills and knowledge develop (many courses that utilize pair programming also have additional assignments that are completed individually outside of class). This type of scaffolding-related strategy is supported by substantial theory on student learning and development (e.g., Vygotsky, 1978), but we do not believe it has been explored in prior research, and existing research on pair programming does not provide direct insights about its efficacy. Other lingering questions include the extent to which duration of the pairing affects student outcomes, whether other groupings (such as triads) might mitigate any negative effects, or whether certain types of programming tasks or levels of complexity lend themselves better to the practice of pair programming. In short, although pair programming consists of some key features, no single study can claim to represent the outcomes of all possible implementations of pair programming. The present study adhered to most of the advice in the Williams et al. (2008) guidelines, but we did not follow every aspect of every guideline. As just one example, we chose to assign pairs instead of letting students choose partners (consistent with their sixth guideline), but we did so randomly rather than trying to create pairs that "maximize the chances students will work well together" (p. 449). Unfortunately, previous research has provided conflicting results on the best approach for creating pairs as well as other relevant considerations.

Given the various possible ways to implement pair programming, additional research that facilitates strong causal inferences is needed. By employing more rigorous research designs, future inquiry will be able to focus on uncovering the conditions in which pair programming may be effective or ineffective. Different approaches for implementing pair programming should be studied systematically, ideally within a large-scale study that explores numerous courses at multiple institutions. Moreover, since publication bias in prior research may be a notable problem (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. This work should also explore the processes through which pair programming may affect student outcomes, as this study was only able to shed partial insights for explaining the negative results among White students.

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; Kuppuswami & Vivekanandan, 2004; Nagappan et al., 2003). Collaborative learning may 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, postsecondary computer labs would only need half the number of workstations that have relevant software to accommodate students in pair programming environments. The outcomes may be even further improved by carefully considering how to create pairs and to design paired assignments with appropriate levels of difficulty and complexity. However, if pair programming often has negative effects on subsequent computer science outcomes, then this practice certainly needs to be reconsidered and potentially eliminated. Future research is crucial for making this determination.

Funding

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.

ORCID

Nicholas A. Bowman http://orcid.org/0000-0001-8899-7383
Lindsay Jarra tp://orcid.org/0000-0002-3031-2916
K. C. Culver ://orcid.org/0000-0001-7929-2680
Alberto M. Se http://orcid.org/0000-0002-8886-6559

References

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). 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. https://doi.org/10.2307/20650280

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. https://doi.org/10.1145/1539024.1508923

Barron, B. (2000). Achieving coordination in collaborative problem-solving groups. *Journal of the Learning Sciences*, *9*(4), 403–436. https://doi.org/10.1207/S15327809JLS0904_2

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

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). 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., Rynes, K., Perrault, J., Hay, K., & Haller, S. (2003). Gender differences in computer science students. In *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on computer science education*, Reno, NV (pp. 49–53).

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. https://doi.org/10.1080/08993408.2014.963363

Biggers, M., Brauer, A., & Yilmaz, T. (2008). Student perceptions of computer science: A retention study comparing graduating seniors vs. CS leavers. *ACM SIGCSE Bulletin*, *40*(1), 402–406. https://doi.org/10.1145/1352322.1352274

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. https://doi.org/10.1016/j.infsof.2007.05.006

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. https://doi.org/10.3102/0034654310383047

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). Jossey-Bass. https://doi.org/10.1002/he.20270

Bowman, N. A., Jarratt, L., Culver, K., & Segre, A. M. (2019a). *How prior pair programming experience affects students' pair programming experiences and outcomes* [Paper presentation]. Proceedings of the Annual Conference on Innovation and Technology in Computer Science Education, Aberdeen, Scotland, 170–175.

Bowman, N. A., Jarratt, L., Culver, K., & Segre, A. M. (2019b). *The impact of pair programming on interest, perceptions, and achievement in computer science* [Paper presentation]. Paper

Presented at the Annual Meeting of the American Educational Research Association, Toronto, ON, Canada.

Bowman, N. A., Jarratt, L., Culver, K., & Segre, A. M. (2020). (Mis)match of students' country of origin and the impact of collaborative learning in computer science. In *Proceedings of the Annual Conference of the American Society for Engineering Education*, Montreal, Quebec, Canada (Paper #30150).

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), 1–21. https://doi.org/10.1145/1921607.1921609

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). 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. https://doi.org/10.1016/j.ijhcs.2007.03.005

Cao, L., & Xu, P. (2005). Activity patterns of pair programming. In *Proceedings of the 38th Hawaii international conference on system sciences*. IEEE.

Chang, M. J., Milem, J. F., & Antonio, A. L. (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). Jossey-Bass.

Chaparro, E. A., Yuksel, A., Romero, P., & Bryant, S. (2005). *Factors affecting the perceived effectiveness of pair programming in higher education*. 17th Workshop of the Psychology of Programming Interest Group, Sussex University, June 2005.

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. https://doi.org/10.1037/a0016239

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. https://doi.org/10.1037/bul0000052

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). IEEE.

Cho, J., & Yu, H. (2015). Roles of university support for international students in the United States: Analysis of a systematic model of university identification, university support, and psychological well-being. *Journal of Studies in International Education*, *19*(1), 11–27. https://doi.org/10.1177/1028315314533606

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. https://doi.org/10.1016/j.infsof.2007.11.002

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. https://doi.org/10.1016/j.jss.2013.12.037

Desjardins, M. (2015, October 22). The real reason U.S. students lag behind in computer science. Fortune. 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. https://doi.org/10.1109/MS.2007.158

Estrada, M., Burnett, M., Campbell, A. G., Campbell, P. B., Denetclaw, W. F., Gutierrez, C. G., Hurtado, S., John, G. H., Matsui, J., McGee, R., Okpodu, C. M., Robinson, T. J., Summers, M. F., Werner-Washburne, M., & Zavala, M. (2016). Improving underrepresented minority student persistence in STEM. *CBE—Life Sciences Education*, *15*(3), es5. https://doi.org/10.1187/cbe.16-01-0038

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. https://doi.org/10.1073/pnas.1319030111

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. https://doi.org/10.1080/08993408.2011.579808

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

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. https://doi.org/10.1016/j.infsof.2009.02.001

Harper, S. R. (2010). An anti-deficit achievement framework for research on Students of Color in STEM. In S. R. Harper & C. B. Newman (Eds.), *Students of Color in STEM* (New Directions for Institutional Research, no. 148, pp. 63–74). Jossey-Bass. https://doi.org/10.1002/ir.362

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

Ho, C. W., Slaten, K. M., Williams, L. A., & Berenson, S. B. (2004). *Examining the impact of pair programming on female students* [NCSU CSC Technical Report 2004-20]. Department of Computer Science, North Carolina State University.

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

Howard, E. V. (2006). Attitudes on using pair-programming. *Journal of Educational Technology Systems*, *35*(1), 89–103. https://doi.org/10.2190/5K87-58W8-G07M-2811

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* (Vol. 27, pp. 41–122). Springer.

Kenny, D. A. (2018). *Mediation*. http://davidakenny.net/cm/mediate.htm.

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. https://doi.org/10.1016/j.edurev.2013.02.002

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). IEEE.

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

Leslie, S.-J., Cimpian, A., Meyer, M., & Freeland, E. (2015). Expectations of brilliance underlie gender distributions across academic disciplines. *Science (New York, N.Y.)*, *347*(6219), 262–265. https://doi.org/10.1126/science.1261375

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

Luo, Z., Wu, S., Fang, X., & Brunsting, N. (2019). International students' perceived language competence, domestic student support, and psychological well-being at a US university. *Journal of International Students*, *9*(4), 954–971. https://doi.org/10.32674/jis.v0i0.605

Malcom, S., & Feder, M. (Eds.). (2016). *Barriers and opportunities for 2-year and 4-year STEM degrees: Systemic change to support students' diverse pathways*. 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*. 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. https://doi.org/10.1145/1145287.1145293

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. https://doi.org/10.1037/dhe0000022

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 [Paper presentation]. Proceedings of the 11th annual SIGCSE conference on innovation and technology in computer science education, Bologna, Italy (pp. 108–112). https://doi.org/10.1145/1140124.1140155

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. https://doi.org/10.1111/j.1467-9280.2007.01995.x

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

Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C., & Balik, S. (2003). Improving the CS1 experience with pair programming. *ACM SIGCSE Bulletin*, *35*(1), 359–362. https://doi.org/10.1145/792548.612006

Nosek, J. T. (1998). The case for collaborative programming. *Communications of the ACM*, *41*(3), 105–108. https://doi.org/10.1145/272287.272333

Pedersen, P., Lonner, W. J., Draguns, J. G., Trimble, J. E., & Scharron-del Rıo, M. R. (Eds.). (2016). *Counseling across cultures* (7th ed.). SAGE.

Pettigrew, T. F., & Tropp, L. R. (2006). A meta-analytic test of intergroup contact theory. *Journal of Personality and Social Psychology*, *90*(5), 751–783. https://doi.org/10.1037/0022-3514.90.5.751

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). IEEE.

Preszler, R. W. (2009). Replacing lecture with peer-led workshops improves student learning. *CBE Life Sciences Education*, *8*(3), 182–192. https://doi.org/10.1187/cbe.09-01-0002

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. https://doi.org/10.1111/j.1365-2729.2007.00224.x

Raudenbush, S. W., & Bryk, A. S. (2002). *Hierarchical linear models: Applications and data analysis methods* (2nd ed.). 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. https://doi.org/10.1109/TSE.2010.59

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. https://doi.org/10.1080/00221546.2016.1257306

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. https://doi.org/10.1007/s10664-008-9093-5

Simon, R. M., Wagner, A., & Killion, B. (2017). Gender and choosing a STEM major in college: Feminity, masculinity, chilly climate, and occupational values. *Journal of Research in Science Teaching*, *54*(3), 299–323. https://doi.org/10.1002/tea.21345

Sinclair, J., & Kalvala, S. (2015). Exploring societal factors affecting the experience and engagement of first year female computer science undergraduates [Paper presentation]. Proceedings of the 15th Koli calling conference on computing education research – Koli Calling '15, Koli, Finland (pp. 107–116). https://doi.org/10.1145/2828959.2828979

Singer, N. (2019, January 24). The hard part of computer science? Getting into class. *The New York Times*. 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.). Sage.

Steele, C. M. (1997). A threat in the air: How stereotypes shape intellectual identity and performance. *The American Psychologist*, *52*(6), 613–629. https://doi.org/10.1037/0003-066X.52.6.613

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

Stinebrickner, R., & Stinebrickner, T. R. (2012). Learning about academic ability and the college dropout decision. *Journal of Labor Economics*, *30*(4), 707–748. https://doi.org/10.1086/666525

Thomas, L., Ratcliffe, M., & Robertson, A. (2003). Code warriors and code-a-phobes: A study in attitude and pair programming [Paper presentation]. Proceedings of the 34th SIGCSE technical symposium on computer science education (pp. 363–367).

Tomcho, T. J., & Foels, R. (2012). Meta-analysis of group learning activities: Empirically based teaching recommendations. *Teaching of Psychology*, *39*(3), 159–169. https://doi.org/10.1177/0098628312450414

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), 11–16. 13. https://doi.org/10.1145/2996201

VanDeGrift, T. (2004). Coupling pair programming and writing: Learning about students' perceptions and processes. *ACM SIGCSE Bulletin*, *36*(1), 2–6. https://doi.org/10.1145/1028174.971306

Varma, R. (2006). Making computer science minority-friendly. *Communications of the ACM*, *49*(2), 129–134. https://doi.org/10.1145/1113034.1113041

Vee, A. (2013). Understanding computer programming as a literacy. *Literacy in Composition Studies*, *1*(2), 42–64. https://doi.org/10.21623/1.1.2.4

Vitores, A., & Gil-Juarez, 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. https://doi.org/10.1080/09589236.2015.1087309

Vygotsky, L. S. (1978). *Mind in society: The development of higher psychological processes*. Harvard University Press.

Walton, G. M., & Cohen, G. L. (2003). Stereotype lift. *Journal of Experimental Social Psychology*, *39*(5), 456–467. https://doi.org/10.1016/S0022-1031(03)00019-2

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., & Kessler, R. (2002). *Pair programming illuminated*. Addison-Wesley.

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

Williams, L., McCrickard, D. S., Layman, L., & Hussein, K. (2008). Eleven guidelines for implementing pair programming in the classroom. In *AGILE 2008 (AGILE'08)* (pp. 445–452). IEEE Computer Society Press.

Williams, L., McDowell, C., Nagappan, N., Fernald, J., Werner, L. (2003, September). Building pair programming knowledge through a family of experiments. In *2003 International symposium on empirical software engineering, 2003. ISESE 2003. Proceedings* (pp. 143–152). IEEE.

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

Williams, L., Wiebe, E., Yang, K., Ferzli, M., & Miller, C. (2002). In support of pair programming in the introductory computer science course. *Computer Science Education*, *12*(3), 197–212. https://doi.org/10.1076/csed.12.3.197.8618

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions. Series A, Mathematical, Physical, and Engineering Sciences*, *366*(1881), 3717–3725. https://doi.org/10.1098/rsta.2008.0118

Wong, B. (2015). Careers "from" science but not "in" science: Why are aspirations to be a scientist challenging for minority ethnic students? *Journal of Research in Science Teaching*, *52*(7), 979–1002. https://doi.org/10.1002/tea.21231

Yadav, A., Zhou, N., Mayfield, C., Hambrusch, S., & Korb, J. T. (2011). Introducing computational thinking in education courses [Paper presentation]. Proceedings of the 42nd ACM technical symposium on computer science education – SIGCSE '11, Dallas, TX (p. 465). https://doi.org/10.1145/1953163.1953297

Yao, C. W., George Mwangi, C. A., & Malaney Brown, V. K. (2019). Exploring the intersection of transnationalism and critical race theory: A critical race analysis of international student experiences in the United States. *Race Ethnicity and Education*, *22*(1), 38–58. https://doi.org/10.1080/13613324.2018.1497968

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. https://doi.org/10.1109/TE.2010.2048328