By Charlie McDowell, Linda Werner,
Heather E. Bullock, and Julian Fernald

Pair Programming Impr Retention, Confidence,

Pair programming produces more proficient, confident programmers—and may help increase female representation in the field.

In recent years, the growth of extreme programming (XP) has brought considerable attention to collaborative programming. Developed over a 15-year period by Kent Beck and his colleagues, Ron Jeffries and Ward Cunningham [1], XP is a computer software development approach that credits much of its success to the use of pair programming by all programmers, regardless of experience [9]. The pair programming dimension of XP requires that teams of two programmers work simultaneously on the same design, algorithm, code, or test. Sitting shoulder to shoulder at one computer, one member of the pair is the "designated driver," actively creating code and controlling the keyboard and mouse. The "non-driver" constantly reviews the keyed data in order to identify tactical and strategic deficiencies, including erroneous syntax and logic, misspellings, and implementations that don't map to the design. After a designated period of time, the partners reverse roles. Code produced by only one partner is discarded, or reviewed collaboratively before it is integrated.

Women and minorities continue to be underrepresented in computer science, and the number of women pursuing college majors in this area is declining. In 1985, 37% of computer science bachelor's degrees were awarded to women; in 2001 that percentage was down to 28% [11]. A number of variables have been proposed to account for this gender difference, including traditional socialization practices that reinforce math and science as male domains, lower confidence ratings and greater math anxiety among women, and women's tendency to take fewer advanced mathematics courses. The belief that computer science is a competitive, alienating field may further discourage women from pursuing careers in this area [5].

Pair programming, when used as a form of collaborative learning, has been shown to increase the number of women (and men) persisting in their previously stated intent to pursue degrees in computer science. In addition, paired teams have been found to significantly outperform individual programmers in terms of program functionality and readability, to report greater satisfaction with the problem-solving process, to have greater confidence in their solutions, and to be more

OVES STUDENT AND PROGRAM QUALITY

likely to complete a programming assignment [10]. Nevertheless, many instructors continue to require students to complete programming assignments independently. Presumably, continued reliance on solo programming in academic settings is rooted in instructor concern that at least one of the partners in a pair will not learn as much as if he or she completed the assignment alone. In the worst case, one member of the pair might do essentially all of the work. Although this would not be "pair programming," it is often difficult, if not impossible, to monitor how students actually spend their programming time and how closely they are following the pairing protocol.

A STUDY OF PAIR PROGRAMMING

We investigated the effects of pair programming on student performance and subsequent pursuit of computer science-related degrees among both female and male college students taking an introductory programming course designed for computer science-related majors (computer science, computer engineering, and information systems management). We collected data on 554 students who attempted the course at the University of California-Santa Cruz [4]. Data was collected from a total of four sections of the course: Fall 2000, Winter 2001 (two sections), and Spring 2001. One of the principle investigators of this study, Charlie McDowell, taught the Fall and Spring sections of the course. The Winter 2001 sections were taught by UCSC faculty members not associated with this project.

Students in the spring section were required to complete programming assignments independently. Students enrolled in the other sections were required to complete all assignments using pair programming. On the first day of class, students in the pairing sections were given a brief 15 to 20 minute description of pair programming and instructed to read Williams and Kessler's article "All I Really Need to Know about Pair Programming I Learned in Kindergarten" [7]. As an incentive they were told the first quiz might include a question on the article.

Students in the pairing sections submitted a list of three names of potential partners, and partners were assigned based on these preferences. In nearly all instances, students

Among students who completed the class, those who paired produced significantly better programs than those who worked alone.

were assigned a partner from their list. Those who stated no preference were randomly assigned a partner. Whenever possible students remained with the same partner throughout the quarter, however, due to schedule changes and drops, a small number of partner reassignments were necessary. As a result of hardships such as heavy work schedules or living far from campus, 17 students across the three pairing sections were permitted to program alone for various reasons. Data from these students was combined with the data from the students in the non-pairing section.

Although each student was assigned to one 90-minute lab time per week, most programming assignments were completed outside of scheduled lab time. The labs functioned primarily as teaching assistant office hours. There were no specific in-lab assignments and attendance was not mandatory. Programming assignments were scored for functionality and readability. Along with each assignment, students submitted a log indicating the amount of time they spent on the assignment (pairing students were asked to differentiate between time spent driving, reviewing, and alone), their level of confidence in their solution, how much they enjoyed working on the assignment, and how satisfied they were with the process.

Regardless of whether they completed assignments in pairs, all students took exams independently. The final exam assessed students' knowledge of programming concepts and their ability to write new code. We collected information about students' SAT scores, the courses they took over the following year, and their major declarations a year after taking the class.

An important assumption of this study was that all four course sections were similar in terms of students' academic preparation to succeed. We found no difference in the SAT math scores among the four sections. We did find that the average SAT verbal score for one of the three pairing sections was lower than the score for the other two pairing sections. However, the difference was not significant when compared to the non-pairing section nor was there a significant difference between the pairing sections as a group and the non-pairing section. Because the difference was only between pairing sections it seemed acceptable.

One of the key hypotheses tested by our study was the following:

Women who program in pairs will have higher retention rates than women who program independently.

Specifically we wanted to know if using pairing as a learning tool for beginning programmers would influence course completion rates and subsequent computer programming course-taking behavior, both in terms of attempts and pass rates, and students' decisions to major in computer science-related fields.

A comparison of students who used pair programming with those who didn't indicated that pairers were significantly more likely to remain in the course through the final exam (90.8%) than were non-pairers (80.4%). Among just those who took the final exam, the difference in pass rates between pairing

(79.6%) and non-pairing students (78.2%) was not statistically significant. Williams and Kessler have proposed "pair pressure" as a possible explanation for higher completion rates among paired versus unpaired students [8]. According to Williams and Kessler, students who work in pairs may be more likely to complete programming courses because of the shared responsibility that results from collaborative partnerships. As a consequence, paired students may remain in the class for the sake of their partner. Although this is a plausible explanation, it is not supported by this data. The fact that in our study there was no difference in pass rates between pairers who completed the course and non-pairers who completed the course suggests that it was not simply the case that pairers were more likely to "stick it out," but rather a larger proportion of paired students were able to master enough of the course material to pass.

We followed those students who passed the introductory programming course for one full academic year beyond the intro course. Consequently this analysis was limited to the 321 students still enrolled at UCSC three quarters after taking and passing the intro course.

Among the students intending to pursue a computer science-related major at the start of the introductory programming class, successfully passed the class with a "C" or better, and were still enrolled at UCSC a full year later (N=238; 187 men and 51 women), a significantly higher percentage of the students who had paired had gone on to attempt the subsequent programming course (Introduction to Data Structures) within a year (84.9%), than had the non-pairing students (66.7%). Separate analyses by gender of the effect of pairing on whether the subsequent course was attempted within a year revealed about an 18% difference between pairers and non-pairers for both women and men (73.8% of paired women vs. 55.6% of non-paired women, and 88.0% of paired men vs. 69.4% of unpaired men). The pairing effect was statistically significant for men but not for women. The increase in the percentage of students associated with pairing appears to be quite similar for men and women. The fact that this difference was statistically significant for men but not women is most likely attributable to the relatively small number of women (51 compared to 186 men) in this part of the study.

Among students who attempted the Data Structures course, the students from the pairing sections were more likely to pass on their first attempt (65.5% vs. 40.0%). That is, students who paired in the introductory programming course were more likely to attempt the subsequent programming class and more likely to pass it than those who learned to program independently. This is particularly significant because students in the Data Structures course were required to complete all programming assignments individually. This indicates that there is not a problem with a significant number of weak students passing the introductory course with the help of their partner, only to fail in the next course where they must program alone. The difference in pass rates between pair and non-pair students was similar for men (64.6% vs. 37.5%) and women (68.3% vs. 44.4%), although for the women the difference was not statistically significant.

	Female		Male		All	
	Pair	Solo	Pair	Solo	Pair	Solo
% that persisted in the course and took the final	88.1	79.5	91.7	81.5	90.8	80.4
% of students taking the final that passed the class with C or better	74.2	74.2	81.3	79.5	79.6	78.2
% of passers that took the 2nd programming course within 1 year	61.1	50.0	81.2	66.1	76.7	62.2
% of passers that took the 2nd course within 1 year—restricted to those indicating a planned CS related major at start of intro course	73.8	55.6	88.0	69.4	84.9	66.7
% of those taking the 2nd course that passed it on the first attempt	68.3	44.4	64.6	37.5	65.5	40.0
% of passers still at UCSC 1 year later that declared a CS major	46.3	11.1	59.5	41.1	56.9	33.8
% of passers still at UCSC 1 year later that declared a CS major—restricted to those indicating a planned CS related major at start of intro course	59.5	22.2	74.0	47.2	70.8	42.2

Shaded numbers indicate statistically significant differences.

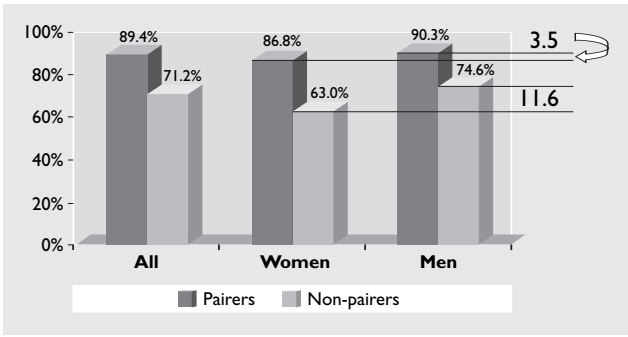
Comparison of completion rates, pass rates, and persistence in the major.

Among the students initially intending a computer science major, and who passed the introductory course and remained at UCSC for at least a year, the pairing students were also more likely to have declared a computer science-related major one year after completing the introductory programming class. This was the case for both women and men. Women who paired were more likely than women who worked independently to be in a computer science related major (59.5% vs. 22.2%). Similarly, pairing men were also more likely to have declared a computer science related major one year later than men who worked alone (74% vs. 47.2%).

Interestingly, the same pattern of results was observed among all students who successfully completed the introductory programming course and were still enrolled at UCSC a year later, regardless of whether they had initially been planning to major in one of the computer science-related majors. Pairers

were significantly more likely to have declared a computer science related major than non-pairers (56.9% vs. 33.8%), and that was the case for both men (59.5% vs. 41.1%), and women (46.3% vs. 11.1%). See the table here for results.

Course performance. In addition to completion and pass rates, we looked at specific parts of the course performance. We measured two related, but distinct indicators of course mastery. The first, the quality of student programs, was operationalized as students' normalized average score on the graded programming assignments. For the second indicator, the extent to which students could apply the concepts covered during the course, we used final exam scores, which all students took independently regardless of whether they paired or not. For the following analyses we included only the 486 students who completed the course. Completing the course is defined as taking the final exam.



Confidence in program solutions—closing one gender gap.

Among students who completed the class, those who paired produced significantly better programs (86.6%) than those who worked alone (68.1%). There was no significant gender difference in average programming scores (men's and women's scores were 81.9% and 82.5% respectively), nor was there an interaction between gender and pairing. In other words, pairing was associated with significantly higher scores for both women and men.

It may be the reluctance of some computer science faculty to use pair programming in their classes is due to a concern that at least some students will "earn" grades that predominantly reflect their partner's work. It is possible, for example, that the pairing students in our study earned higher average programming scores simply because weaker students received scores that were primarily due to the work of the stronger student in the pair, thus artificially inflating the average programming scores of the pairers.

Elsewhere we have argued that the very process of working collaboratively enhances the quality of programs that pairs produce [3]. In that paper we compared two sections of an introductory programming course taught by the same instructor, and for which assignments were intentionally designed to be equivalent. We found the average score on programming assignments of students in the pairing section was significantly higher than the average score of the top 50% (based on final exam scores) of the non-pairing section.

Because students in different sections of this study did not complete exactly the same programming assignments, we did a follow-up study [2] in which a section of pairing students was given the same programming assignments as the non-pairing students from our original study. In that study we again found that the programs produced by the pairing students were significantly better than those produced by the non-pairing students, although the difference was not as great as in the original study. This suggests that some of the difference reported in our main study could be from variations in the difficulty of the assignment, but that the overall conclusion is unchanged.

Because all of the students in this study took the final exam independently, we considered final exam scores to be a strong indication of the extent to which students had mastered the course material. There was no significant difference in the average final exam of the pairers (75.2%) and the non-pairers (74.4%). This finding strongly suggests a student's ability to independently apply concepts to novel problems is not compromised by learning to program in pairs. Indeed, considering that a significantly greater percentage of the students who paired took the final, it seems that learning to program in pairs results in mastery for a greater percentage of students.

Confidence and enjoyment. Of course the most important goal for students in any class is mastery of the material. This is certainly the case for introductory programming courses, where future success is dependent on a strong foundational knowledge. However, subjective experiences in introductory programming courses may also contribute to decisions about whether to pursue computer science-related degrees. For this reason it is important to understand how the experience of pairing influences students' confidence and enjoyment of their work. Students responded to the following questions in their logs completed after each graded programming assignment.

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

Enjoyment: How much did you enjoy working on this programming assignment? (1=not at all, 7=very much)

Paired students enjoyed working on programming assignments (M=5.15) more than non-pairing students (M=4.69), Likewise, among just the males, paired students reported greater enjoyment (M=5.23) than non-pairing students (M=4.75). Women paired students also reported greater enjoyment (M=4.90) than non-pairing women (M=4.65), however this difference was not significant. There was also no significant difference in the reported enjoyment of all women versus all men.

In addition to enjoying their coursework more, students who paired reported significantly higher confidence in their program solutions (89.4%) than students who worked independently (71.2%). Consistent with findings in other areas, men were significantly more confident (87.0%) than women (81.1%). There was also a significant interaction between pairing and gender with regard to reported confidence. Follow-up tests of the interaction indicated that pairing resulted in more confidence for both women (86.8% vs. 63.0%) and men (90.3% vs. 74.6%). However, the 24% increase in confidence that pairing afforded women was even greater than the 15% confidence boost experienced by men who had the benefit of pairing. The result was a significant decrease of a gender gap in confidence as shown in the figure here.

Conclusion

The results of this study provide some of the most compelling evidence to date of the effectiveness of pair programming as a pedagogical tool. It appears that pairing bolsters course completion and consequently course pass rates, and contributes to greater persistence in computer science-related majors. Moreover, students who paired were more likely to pass the subsequent programming course that required them to work alone. This is a strong indicator that pairing did not result in a significant number of students passing the course without learning how to program due to a "free ride" from their partner. The pairing students also produce higher quality programs, are more confident in their work, and enjoy it more. We hope these findings will encourage instructors to use pair programming not only in their introductory courses, but also in their upper-level courses.

The continued underrepresentation of women in computer science underscores the need for strategies that foster women's interest and promote their success [5]. Pair programming appears to be one such approach [6]. That the benefits associated with pair programming extend to both men and women speaks to its broad-based appeal. As we continue to investigate the effects of this technique on attracting and retaining female students, parallel research investigating these phenomena in the workplace is also needed. C

This work was funded by National Science Foundation grant EIA-0089989. Any opinions, findings, and conclusions or recommendations expressed in this article are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

1. Beck, K. Extreme *Programming Explained: Embrace Change.* Addison-Wesley, Reading, MA, 2000.
2. Hanks, B. and McDowell, C. Program quality with pair programming in CS1. In *Proceedings of the 9th Annual Conference on Innovation and Technology in Computer Science Education.* (Leeds, UK, 2004), SIGCSE Bulletin, 176–180.
3. McDowell, C., Werner, L., Bullock, H., and Fernald, J. The effects of pair-programming on performance in an introductory programming course. In *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education* (KY, 2002) 38–42.
4. McDowell, C., Werner, L Bullock, H., and Fernald, J. The impact of pair programming on student performance and pursuit of computer science related majors. In *Proceedings of the 25th International Conference on Software Engineering. IEEE Computer Society* (Portland, OR, 2003), 602–607.
5. *Tech-Savvy Educating Girls in the New Computer Age*. 2000. American Association of University Women Education Foundation; Executive summary at www.aauw.org/research/techexecsumm.cfm.
6. Werner, L.L., Hanks, B., and McDowell, C. Pair-programming helps female computer science students. *J. Educational Resources in Computing 4,* 1 (2005).
7. Williams, L.A. and Kessler, R.R. All I really need to know about pair programming I learned in kindergarten. *Commun. ACM 43*, 5 (May 2000), 108–114.
8. Williams, L.A. and Kessler, R.R. The effects of "pair-pressure" and "pair-learning" on software engineering education. In *Proceedings of the 13th Conference on Software Engineering Education and Training. IEEE Computer Society* (Austin, TX, 2000), 59–65.
9. Williams, L., Kessler, R., Cunningham, W., and Jeffries, R. Strengthening the case for pair programming. *IEEE Software 17*, 4 (2000), 19–25.
10. Williams, L., McDowell, C., Nagappan, N., Fernald, J., and Werner, L.L. Building pair programming knowledge through a family of experiments. In *Proceedings of the IEEE International Symposium on Empirical Software Engineering.* (Rome, Italy, 2003), 143–153.
11. Women, Minorities and Persons with Disabilities in Science and Engineering. NSF. 2004; www.nsf.gov/statistics/women.

Charlie McDowell (charlie@cs.ucsc.edu) is a professor in the Computer Science Department at the University of California, Santa Cruz.
Linda Werner (linda@cs.ucsc.edu) is a lecturer in the Computer Science Department at the University of California, Santa Cruz.
Heather E. Bullock (hbullock@cats.ucsc.edu) is an associate professor in the Psychology Department at the University of California, Santa Cruz.
Julian Fernald (jfernald@cats.ucsc.edu) is the director of Institutional Research in the Psychology Department at the University of California, Santa Cruz.