Pair Programming and Gender

Linda L. Werner
University of California, USA

Brian Hanks
Fort Lewis College, USA

Charlie McDowell
University of California, USA

INTRODUCTION

Studies of pair programming both in industry and academic settings have found improvements in program quality, test scores, confidence, enjoyment, and retention in computer-related majors. In this article we define pair programming, summarize the results of pair programming research, and show why we believe pair programming will help women and men succeed in IT majors.

BACKGROUND

Traditional undergraduate introductory programming courses generally require that students work individually on their programming assignments. In these courses, working with another student on programming homework constitutes cheating and is not tolerated. The only resources available to help students with problems are the course instructor, the textbook, and the teaching assistant. They are not allowed to work with their peers, who are struggling with the same material. This pedagogical approach teaches introductory programming students that software development is an individual activity, potentially giving students the mistaken impression that software engineering is an isolating and lonely career. Gender studies suggest that such a view will disproportionately discourage women from pursuing IT careers (Margolis & Fisher, 2002).

Cooperative or collaborative learning models involve two or more individuals taking turns helping one another learn information (Horn, Collier, Oxford, Bond, & Dansereu, 1998). Some researchers differentiate between cooperative and collaborative methods by stating that cooperative learning involves students taking responsibility for subtasks, whereas collaborative learning requires that the group works together on all aspects of the task (Underwood & Underwood, 1999). The consensus from numerous field and laboratory investigations is that academic achievement such as performance on a test is enhanced when an individual learns information with others as opposed to individually (O'Donnell & Dansereu, 1992; Slavin, 1996; Totten, Sills, & Digby, & Russ, 1991).

Cooperative activities have been taught and practiced for other software system development tasks such as design and software engineering but not for programming (Basili, Green, Laitenburger, Lanubile, Shull, Sorumgard, et al., 1996; Fagan, 1986, Sauer, Jeffrey, Land, & Yetton, 2000; Schlimmer, Fletcher, & Hermens, 1994). Often cooperative methods are used in upper division computer science (CS) courses such as compiler design and software engineering in which group projects are encouraged or required. In these courses, the group projects are split up by the group members and tackled individually before being recombined to form a single solution. Sometimes a software engineering instructor offers assistance to the student groups regarding techniques for cooperation but these topics are rarely discussed in other CS courses.

The benefits of collaboration while programming in both industrial and academic settings have been discussed by Flor and Hutchins (1991), Constantine (1995), Coplien (1995), and Anderson, Beattie, Beck, Bryant, DeArment, Fowler, Fronczak, et al. (1998). However, the recent growth of extreme programming (XP) (Beck, 2000) has brought considerable attention to the form of collaborative programming

known as pair programming (Williams & Kessler, 2003). Extreme programming is a software development method that differs in a number of ways from generally accepted prior software development methods. These differences include writing module tests before writing the modules, working closely with the customer to develop the specification as the program is developed, and an emphasis on teamwork as exemplified by pair programming, to name just a few. The emphasis on teamwork is an aspect of extreme programming that may be particularly appealing to women.

With pair programming, two software developers work side-by-side at one computer, each taking full responsibility for the design, coding, and testing of the program under development. One person is called the driver and controls the mouse and keyboard; the other is called the navigator and provides a constant review of the code as it is produced. The roles are reversed periodically so that each member of the pair has experience as the driver and navigator. Studies have shown that pair programming produces code that has fewer defects and takes approximately the same total time as when code is produced by a solitary programmer (Nosek, 1998; Williams, Kessler, Cunningham, & Jeffries, 2000). Any code that is produced by only one member of a pair is either discarded or reviewed by the pair together prior to inclusion into the program.

PAIR PROGRAMMING IN THE CLASSROOM

Early experimental research with pair programming using small numbers of students or professional programmers found that pairs outperformed those who worked alone (Nosek, 1998; Williams et al., 2000). Pairs significantly outperformed individual programmers in terms of program functionality and readability, reported greater satisfaction with the problem-solving process, and had greater confidence in their solutions. Pairs took slightly longer to complete their programs, but these programs contained fewer defects.

A series of experiments conducted at the University of California at Santa Cruz (UCSC) (Hanks, McDowell, Draper, & Krnjajic, 2004; McDowell, Werner, Bullock, & Fernald, 2003a; Werner, Hanks, & McDowell, 2005) found that students who pair programmed in their introductory programming course were more confident in their work. They were also more likely to complete and pass the course, to take additional computer science courses, to declare computer-related majors, and to produce higher quality programs than students who programmed alone.

Naggapan, Williams, Ferzli, Yang, Wiebe, Miller, et al. (2003) report that pair programming results in programming laboratories that are more conducive to advanced, active learning. Students in these labs ask more substantive questions, are more productive, and are less frustrated.

To ensure that paired students enjoy these benefits, it is important that they have compatible partners. Researchers at the University of Wales (Thomas, Ratcliffe, & Robertson, 2003) investigated issues regarding partner compatibility for pair programming students. They asked more than 60 students to indicate their self-perceived level of expertise and confidence in their programming abilities, and used these rankings to evaluate pairing success. It is important to note that self-reported ability and actual ability are different measures, as 5 of the 17 students who felt that they were highly capable did very poorly in the course.

Thomas et al. found some evidence that students do their best work when paired with students with similar confidence levels. Students with less self-confidence seem to enjoy pair programming more than those students who reported the highest levels of confidence. As there were only seven women in the class, no conclusions about how pairing affected them can be made.

Researchers at North Carolina State University investigated factors that could affect student pair compatibility. Out of 550 graduate and undergraduate students, more than 90% reported being compatible with their partner (Katira, Williams, Wiebe, Miller, Balik, & Gehringer, 2004). Factors such as personality type, actual skill level, and self-esteem appear to have little, if any, effect on partner compatibility. The authors do not discuss any relation between gender and compatibility. Students reported they were more compatible with partners who they perceived to have similar levels of technical competence as themselves; unfortunately, there is no pro-

active way that instructors can use this factor to assign partners.

The benefits of pair programming are enjoyed by all students, but women students appear to benefit more. Research conducted at UCSC (McDowell, Werner, Bullock, & Fernald, 2003, Werner, Hanks, & McDowell, 2005a) indicates that although students who pair are more confident in the work and are more likely to declare computer-related majors, these increases are greater for women than men.

The UCSC study looked at four sections of an introductory programming course over three academic terms, one in the fall, two in the winter, and one in the spring. The fall and winter sections required pair programming; the spring term required students to work alone. The fall and spring sections were taught by the same instructor; the winter sections were taught by two additional instructors. Students enrolled into sections without knowing about the pair programming experiments. There were no differences in SAT Math scores or high school GPA between the pairing and non-pairing groups.

Among the UCSC introductory programming students who indicated intent to declare a computer-related major, 59.5% of the paired women had declared a CS-related major one year later, compared with only 22.2% of the women who worked alone. Men who paired were also more likely to declare a CS-related major within one year of taking the introductory course; 74.0% of the paired men had declared a CS-related major compared with 47.2% of the men who worked alone. This is an instance of a possible positive impact on the gender gap due to pair programming. Without pairing, men are 2.13 times more likely than women to declare a CS-related major. When pairing is used, men are only 1.24 times more likely than women to declare a CS-related major.

In this same study, the confidence of students was also measured. To assess student confidence levels, students in the study responded to the question, "On a scale from 0 (not at all confident) to 100 (very confident), how confident are you in your solution to this assignment?" when they turned in each of their programming assignments.

Overall, students who were paired reported significantly higher confidence in their program solutions (89.4) than students who worked independently (71.2). Although men as a group were significantly more confident (87.0) than women (81.1), there was a significant interaction between pairing and gender with regard to reported confidence. Simple effects follow-up tests of the interaction indicated that pairing resulted in increased confidence for both women (86.8 vs. 63.0) and men (90.3 vs. 74.6). Women's confidence increased by 24 points when they paired compared with a 15-point increase for men. Pairing had a statistically significantly greater effect on confidence levels for women and, therefore, may have a visible positive impact on the gender gap. Unpaired men reported 1.18 times greater confidence than unpaired women, while paired men reported 1.04 times greater confidence than paired women. Pairing seems to close the confidence gap between women and men.

FUTURE TRENDS

The Pipelines column of Computing Research News published a short article on pair programming (Werner, Hanks, McDowell, Bullock, & Fernald, 2005b). Computing Research News is a publication of the Computing Research Association whose mission is to influence policy. The appearance of this topic in a publication directed at computing related department chairs of PhD granting institutions is indicative of its importance to the future of introductory computer programming instruction.

One drawback of pair programming is its collocation requirement. We are investigating techniques for extending pair programming to situations where it is difficult or impossible for students to physically meet (Hanks, 2004, Hanks, 2005).

Recommendations for next steps with this research include: the study of pair programming in high school and middle school, additional study about pair compatibility and what is needed in order to increase performance, and to determine what strategies or instructional support is needed to create effective pairs.

CONCLUSION

Why does pair programming hold promise for closing the gender gap regarding the field of information technology? The American Association of Univer-

sity Women's report in 2000 gives four reasons for the decline in enrollment of women in computer science (CS) programs. These reasons are:

1. A perception that a career in computing is not conducive to family life
2. A belief that work in the information technology field is conducted in a competitive rather than a collaborative environment
3. A perception of CS as a solitary occupation
4. Concern about safety and security about women working alone at night and on weekends in computer laboratories

The use of pair programming combats at least three of these four reasons. A typical beginning programming course requires individual work; with the use of pair programming, women may view programming as a collaborative exercise. Williams and Kessler suggest that "peer pressure" may be at work as a possible explanation for higher completion rates among paired vs. solo programming students (Williams & Kessler 2000). It may be the collaborative aspect of pair programming that is a major reason that the students remain in the class. The increased levels of confidence that can be attributed to pairing are probably also a factor in improved retention. This same collaboration could help combat the perception of CS as a solitary occupation. Additionally, an outcome of pair programming is that no one works alone late at night or on weekends in a computer laboratory. Partners work together. We hypothesize that these reasons cause pair programming to contribute to persistence of women in CS.

REFERENCES

American Association of University Women Education Foundation Commission on Technology, Gender, and Teacher Education. (2000). *Tech-savvy educating girls in the new computer age*. Retrieved January 1, 2005, from http://www.aauw.org/2000/techsavvy.html

Anderson, A., Beattie, R., Beck, K., Bryant, D., DeArment, M., Fowler, M., et al. (1998). *Chrysler goes to extremes, distributed computing* (pp. 24-28). Retrieved January 1, 2005, from http://216.239.63.104/search?q=cache:JdfJA5iROdQJ:www.xprogramming.com/publications/dc9810cs.pdf+&hl=en

Basili, V. R., Green, S., Laitenburger, O., Lanubile, F., Shull, F., Sorumgard, S., et al. (1996). The empirical investigation of perspective-based reading. *Journal of Empirical Software Engineering, 1*(2), 133-164.

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

Constantine, L. L. (1995). *Constantine on peopleware*. Englewood Cliffs, NJ: Yourdon Press.

Coplien, J. O. (1995). A development process generative pattern language. In J. O. Coplien & D. C. Schmidt (Eds.), *Pattern languages of program design* (pp. 183-237). Reading, MA: Addison-Wesley.

Fagan, M. E. (1986). Advances in software inspections. *IEEE Transactions on Software Engineering, 12*(7), 744-751.

Flor, N. V., & Hutchins. E. L. (1991). *Analyzing distributed cognition in software teams: A case study of team programming during perfective software maintenance*. Empirical Studies of Programmers: Fourth Workshop.

Hanks, B. (2004). Distributed pair programming: An empirical study. In *Proceedings of the 4th Conference on Extreme Programming and Agile Methods—XP/Agile Universe* (LNCS No. 3134, pp. 81-91). Springer.

Hanks, B. (2005). Student performance in CS1 with distributed pair programming. To appear in *The Proceedings of the 10th Tenth Annual Conference on Innovation and Technology in Computer Science Education*, Lisbon, Portugal.

Hanks, B., McDowell, C., Draper, D., & Krnjajic, M. (2004). Program quality with pair programming in CS1. In *Proceedings of the 9th Annual SIGCSE conference on Innovation and Technology in Computer Science Education* (pp. 176-180).

Horn, E. M., Collier, W. G., Oxford, J. A., Bond, C. F., & Dansereu, D. F. (1998). Individual differences in dyadic cooperative learning. *Journal of Educational Psychology, 90*(1), 153-160.

Katira, N., Williams, L., Wiebe, E., Miller, C., Balik, S., & Gehringer, E. (2004). On understanding compatibility of student pair programmers. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education* (pp. 7-11).

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

McDowell, C., Werner, L., Bullock, H., & Fernald, J. (2003). The impact of pair programming on student performance, perception, and persistence. In *Proceedings of the 25th International Conference on Software Engineering*, Portland, Oregon (pp. 602-607).

Naggapan, N., Williams, L., Ferzli, M., Yang, K., Wiebe, E., Miller, C., & Balik, S. (2003). Improving the CS1 experience with pair programming. In *Proceedings of the 34th SIGCSE technical symposium on computer science education* (pp. 359-362).

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

O'Donnell, A. M., &. Dansereu, D. F. (1992). Scripted cooperation in student dyads: A method for analyzing and enhancing academic learning and performance. In R. Hartz-Lazarowitz & N. Miller (Eds.), *Interactions in cooperative groups: The theoretical anatomy of group learning* (pp. 120-141). London: Cambridge University Press, 1992.

Sauer, C., Jeffrey, D. R., Land, L., & Yetton, P. (2000). The effectiveness of software development technical review: A behaviorally motivated program of research. *IEEE Transactions on Software Engineering, 26*(1), 1-14.

Schlimmer, J. C., Fletcher, J. B., & Hermens, L.A. (1994). Team-oriented software practicum. *IEEE Transactions on Education, 37*(2), 212-220.

Slavin, R. E. (1996). Research on cooperative learning and achievement: When we know, what we need to know. *Contemporary Educational Psychology, 21*, 43-69.

Thomas, L., Ratcliffe, M., & Robertson, A. (2003). Code warriors and code-a-phobes: A study in attitude and pair programming. In *Proceedings of SIGCSE Technical Symposium on Computer Science Education* (pp. 363-367).

Totten, S., Sills, T., Digby, A., & Russ, P. (1991). *Cooperative learning*. New York: Garland.

Underwood, J., & Underwood, G. (1999). Task effects on cooperative and collaborative learning with computers. In K. Littleton & P. Light (Eds.), *Learning with computers* (pp. 10-23). New York: Routledge.

Werner, L., Hanks, B., & McDowell, C. (2005a). *Female computer science students who pair program persist*. To appear in JERIC.

Werner, L., Hanks, B., McDowell, C., Bullock, H., & Fernald, J. (2005b, March). Expanding the pipeline: Want to increase retention of your female students? *Computing Research News*, p. 2. Retrieved March 21, 2005, from http://www.cra.org/CRN/issues/0502.pdf

Williams, L. A., & Kessler, R. R. (2000). The effects of "pair-pressure" and "pair-learning" on software engineering education. *The 13th Conference on Software Engineering Education and Training*. Austin, TX: IEEE Computer Soc.

Williams, L., & Kessler, R. (2003). *Pair programming illuminated*. Addison-Wesley.

Williams, L., Kessler, R., Cunningham, W., & Jeffries, R. (2000, July/August). Strengthening the case for pair-programming. *IEEE Software, 17*.

KEY TERMS

Active Learning: Based on a theory of learning where people are able to set goals, plan, and revise. This is to be contrasted with the theory of learning based on Piaget where newborns are born with a blank slate on which learning is placed.

Collaborative Learning: A learning model where students work together in a group and the group works together on all aspects of the task.

Cooperative Learning: A learning model where students work together in a group but individual students take responsibility for various subtasks.

Extreme Programming: A method of software development that emphasizes customer involvement and teamwork. One component of the teamwork is the use of pair programming for all code development.

Module: An independent part of a computer program. Different computer languages typically have their own terminology for module. Some of these are: function, procedure, subroutine, and method.

Pair Programming: A method of software development where two software developers work side-by-side at one computer, each taking full responsibility for the design, coding, and testing of the program under development. One person is called the driver and controls the mouse and keyboard; the other is called the navigator and provides a constant review of the code as it is produced.

Software Development: The development of software goes through the steps of requirements specification, design, code, test, and maintenance. The development method that is used determines the order, length, and specific details for each of these steps.