Pair Programming and Gender

Linda L. Werner
University of California, Santa Cruz, USA

Brian Hanks
Fort Lewis College, USA

Charlie McDowell
University of California, Santa Cruz, 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, 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, Kessler, Cunningham, & Jeffries, 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 compe-