Debunking the Nerd Stereotype with Pair Programming

Laurie Williams
North Carolina State University

Using pair programming can be a way to attract and retain students in computing majors.

Jamie wants to be a software engineer. She enjoyed her programming and science classes in high school and wants to combine her interest in both disciplines to help society through biomedical applications.

Since she started college, it seems that her life has been centered on time-consuming programming classes. In those classes, her professors insist that she work alone—some professors expressly forbid even discussing assignments with fellow classmates. Before entering college, Jamie was aware of the stereotypical view that programmers work long hours by themselves. Based on her college experience, now she knows it's more than just a stereotype—it's true. Perhaps she should forget programming. She likes the friends she's met in her biology lab group—maybe biology would be a better major.

Jamie's teachers, though, actually aren't giving her an accurate depiction of life as a software engineer. Based on my own industry experience combined with anecdotal evidence and observation, I have the view that a software engineer's daily life involves a significant amount of interpersonal communication and teamwork. I also know that when we ask industry representatives how we can better educate our undergraduates, improving communication and teamwork skills is consistently at the top of their list. So, they must not be hiring our students to work alone all the time.

In early 2005, I conducted an e-mail survey of industry practitioners in software engineering jobs. I asked them what percentage of the day they worked alone versus working with one other person or with more than one other person. The 359 responses from practitioners in 94 companies in 21 different countries included 270 responses from the US.

The results of this survey indicated that, on average, software engineers spend 63 percent of their time working alone, 24 percent working with one other person, and 13 percent working with more than one other person. Yes, 63 percent is a significant amount of time spent working alone, but it is probably commensurate with other professions that don't have a comparable nerdy, work-alone reputation, such as lawyers and other engineering fields. And this percentage means that an average of three hours of a software engineer's eight-hour workday involves collaboration, teamwork, and communication.

In many software engineering curricula, students work alone for the first year or two. Then, later courses incorporate more teamwork and collaboration. But the fact is that by then many students have switched to other majors.

By allowing more collaboration in our software engineering classes, specifically in the form of pair programming, I believe we can begin to debunk the "nerd myth" that programmers work alone all the time—a misconception that could deter many students from entering the computing field.

LET ME WORK WITH FRIENDS

To increase and broaden participation in computing, we can appeal to the desire of millennial students (those born after 1982),[1-3] women,[4] and some underrepresented groups[5] to learn and work in a collaborative environment.

Almost one-half of computer science students we have studied are Myers-Briggs extroverts.[6] Millennial students, who are the majority of current college enrollees, show strong learning preferences toward teamwork, experiential learning, and a collaborative style of working.[1,2] They are used to being organized in teams with people they "click" with to provide for social aspects of work.[3] They learn most effectively through discovery, and they prefer to learn in a participatory manner rather than by being told what to do.[2]

The success rate of underrepresented minorities in science courses has been shown to be dramatically improved by shifting the learning paradigm from individual study to one that capitalizes on group processes,

such as student work groups and student-student tutoring.[7]

In an experiment with calculus students, Uri Treisman[5] found that about 60 percent of African-American and Hispanic students who had completed calculus at Berkeley in the preceding decade received grades of D or F, grades so low that they could not proceed with a major in mathematics, science, or technology. Treisman's research revealed that prior social norms had taught these minority students that they needed to study alone because working with others was cheating. In contrast, the most successful Asian students formed "study squads" to get through calculus, groups in which having the ability to help others increased an individual's social status.

Treisman then told his students they were required to do peer checking of each other's work. The students also worked in collaborative small groups in class. Subsequently, only about 4 percent of the minority students completing Treisman's calculus workshop made a D or F, compared with the earlier 60 percent rate.[8]

PAIR PROGRAMMING BENEFITS

Prior studies indicate that female students can be concerned about the insularity of working alone for long periods of time, as they perceive to be the case with computer science and IT careers.[9,10] Jane Margolis and Allan Fisher[4] cite instances in which undergraduate women's confidence in their ability to succeed in computer science declines when they feel they are spending long hours on assignments that fellow students appear to finish in comparatively little time.

Pair programming can help prevent this confidence degradation in two ways. First, the participants experience how long their fellow students actually work and how much they actually know (or do not know). Second, working together, the pair tends to more easily figure out an assignment and to finish faster.[11,12]

In addition to the practitioner survey, I also conducted a survey of educators on the mailing list for the ACM's Special Interest Group on Computer Science Education (SIGCSE) in which I asked the following questions:

- For the assignments in your computer science classes, what is your policy on students working together?
- What is the rationale behind your policy?
- Have you ever allowed students to do pair programming?
- If so, what class? Have you decided to continue or discontinue the use of pair programming in your class? What made you decide to do so?
- If not, why do you feel students should not pair program?

Many educators cited significant benefits from allowing the students to do pair programming.

Fifty educators—17 from colleges, 32 from universities, and one from a high school—shared their views.

Many cited significant benefits from allowing the students to work together and do pair programming. First, they felt their students demonstrated less anxiety and frustration; the collaboration helped to create a more supportive study environment and more camaraderie in the classroom. Second, they felt the students learned more, produced more, and submitted higher-quality assignments. Finally, the educators indicated that their own workload had decreased. They spent half as much time grading their students' work, uncovered fewer cheating cases, and spent much less time answering students' questions on small-scale technical quandaries, all while spending more time focusing on the big issues.

EXAMINING EDUCATORS' CONCERNS

These same educators expressed some concerns about using the pair programming approach in their classrooms. For example, they commonly indicated that the students should learn the fundamentals on their own and gain confidence in their own abilities prior to being allowed to work with others in the later years of the curriculum.

This type of sentiment drives some academic integrity policy statements such as these found on the Internet in syllabi for beginning computer science classes:

- You must work completely alone on these assignments, except for help from the teaching assistant or instructor.
- Do not discuss the assignment with anyone.
- Do not give or take ideas on how to solve the problem; clever ideas and a good design lead to higher grades on the programs, and those higher grades are only due the person who thought up the design ideas.

Policies such as these can make computer science quite scary for students such as Jamie. In contrast, pair programming can be a way to attract and retain students in computing-related disciplines.

Is it really that important for students to learn the fundamentals alone? A simple, "Hey, you forgot to close your bracket," every now and then from a pairing partner might just make all the difference in the world. Students sometimes feel self-conscious when asking authority figures for help, whereas asking a friend for a few pointers is not only less stressful, but more like a real-life scenario. Pair programming, in fact, helps students learn fundamental skills, rather than impeding that learning.

At North Carolina State University (NCSU) and the University of California, Santa Cruz (UCSC), extensive pair programming studies were conducted with approximately 1,200 beginning computer science students (CS1)[13] and with nearly 300 third- and fourth-year software engineering students[6,14] over a three-year period. According to these studies, students in classes in which pair programming was required generally had higher pro-

ject scores and higher exam scores—a sign that the students are learning the material individually as well. More importantly, more students passed CS1 with a grade of C or better.

At NCSU, when they went on to the second programming class (CS2) in which they were required to work alone—and thereby demonstrate their individual competence—the students who had pair programmed in CS1 were more likely to maintain or improve their grades than the students who had worked alone in CS1.

At UCSC, more students who had paired in CS1 attempted CS2 (77 percent versus 62 percent), and slightly more of these students passed CS2 compared with the students who had worked solo in CS1.

Most importantly, a higher percentage of the students who took a paired CS1 class chose to pursue a computer science-related major one year later compared with their solo counterparts: NCSU—57 percent versus 34 percent, $p < 0.001$; UCSC—25 percent versus 11 percent, $p < 0.008$.

Research has shown a positive correlation between interaction in course instruction and student retention.[2] As educators, our challenge is to adjust our pedagogy to respond to our students' learning styles and preferences.

Our studies show that using pair programming as a structure for incorporating collaboration in the classroom helps increase and broaden participation in computing fields and helps debunk the myth that programmers work alone all the time. It's also a way for students to get a better view of—and feel more confident in—their preparation for working in the real world.

The face of the IT workforce is changing. As the millennial generation makes its way into the working world, the archetype of the nerd as the introverted, obsessed computer programmer must share equal space with the chatty, social software engineer. By showing the millennial generation that computer science isn't all about twiddling bits and squashing bugs, we can go a long way toward diversifying our field and leveraging the strengths of the next generation. ■

Acknowledgments

Many thanks to Michael Wing for his entertaining and informative geek exposé ("In Praise of SE Geeks," *ACM Software Engineering Notes*, July 2005), which inspired me to try to debunk the nerd stereotype rather than the geek stereotype. I also thank my esteemed colleagues who worked with me on the pair programming research reported here: Sally Berenson, Jason Osborne, Lucas Layman, Charlie McDowell, Nachiappan Nagappan, Kelli Slaten, Hema Srikanth, Mladen Vouk, Eric Wiebe, Linda Werner, and Kai Yang. Chris Williams provided the image for this article. National Science Foundation grants ITWF 00305917 and CNS 0540523 provided funding for this research. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

References

1. D. Oblinger, "Boomers, Gen-Xers, and Millennials: Understanding the New Students," *Educause Review*, July/Aug. 2003, pp. 37-47.
2. D. Oblinger and J. Oblinger, eds., *Educating the Net Generation*, Educause, 2005.
3. C. Raines, "Managing Millennials," 2002; www.generationsatwork.com/articles/millenials.htm.
4. J. Margolis and A. Fisher, *Unlocking the Clubhouse: Women in Computing*, MIT Press, 2002.
5. U. Treisman, "Studying Students Studying Calculus: A Look at the Lives of Minority Mathematics Students in College," *The College Mathematics J.*, vol. 23, no. 5, 1992, pp. 362-372.
6. L. Layman, T. Cornwell, and L. Williams, "Personality Types, Learning Styles, and an Agile Approach to Software Engineering Education," *Proc. ACM Technical Symp. Computer Science Education* (SIGCSE 06), ACM Press, to appear, 2006.
7. C.E. Nelson, "Student Diversity Requires Different Approaches to College Teaching, Even in Math and Science," *American Behavioral Scientist*, vol. 40, no. 2, 1996, pp. 165-175.
8. R.E. Fullilove and P.U. Treisman, "Mathematics Achievement among African American Undergraduates of the University of California, Berkeley: An Evaluation of the Mathematics Workshop Program," *J. Negro Education*, vol. 59, no. 3, 1990, pp. 463-478.
9. American Association of University Women Education Foundation, "Educating Girls in the New Computer Age," 2000; www.aauw.org/member_center/publications/TechSavvy/TechSavvy.pdf.
10. P. Freeman and W. Aspray, *The Supply of Information Technology Workers in the United States*, Computing Research Association, 1999.
11. L.A. Williams, "The Collaborative Software Process," PhD dissertation, Dept. Computer Science, Univ. of Utah, Salt Lake City, 2000.
12. L. Williams and R. Kessler, *Pair Programming Illuminated*, Addison-Wesley, 2003.
13. L. Williams et al., "Building Pair Programming Knowledge through a Family of Experiments," *Proc. Int'l Symp. Empirical Software Engineering* (ISESE), 2003, pp. 143-152.
14. K. Slaten et al., "Understanding Student Perceptions of Pair Programming and Agile Software Development Methodologies: Verifying a Model of Social Interaction," *Proc. Agile 2005 Conf.*, 2005, pp. 323-330.

***Laurie Williams** is an assistant professor in the Department of Computer Science at North Carolina State University. Contact her at williams@csc.ncsu.edu.*

Series editor: Juan E. Gilbert, Dept. Computer Science and Software Engineering, Auburn University; bpc@computer.org