DOI:10.1145/2492007.2492020

Leo Porter, Mark Guzdial, Charlie McDowell, and Beth Simon

Education

Success in Introductory Programming: What Works?

How pair programming, peer instruction, and media computation have improved computer science education.

MANY *COMMUNICATIONS* READERS have been in faculty meetings where we have reviewed and bemoaned statistics about how bad attrition is in our introductory programming courses for computer science majors (CS1). Failure rates of 30%–50% are not uncommon worldwide.[1] There are usually as many suggestions for how to improve the course as there are faculty in the meeting. But do we know anything that really works?

We do, and we have research evidence to back it up. Pair programming, peer instruction, and media computation are three approaches to reforming CS1 that have shown positive, measurable impacts. Each of them is successful separately at improving retention or helping students learn, and combined, they have a dramatic effect.

Pair Programming

Pair programming is a practice that started in industry as an agile method. The idea is to have two people at a keyboard, one as the "driver" and the other as an "observer" or "navigator." The two people in the pair swap roles regularly while working. There is significant evidence that having two people at the keyboard improves productivity in industry, but does it help in the classroom?

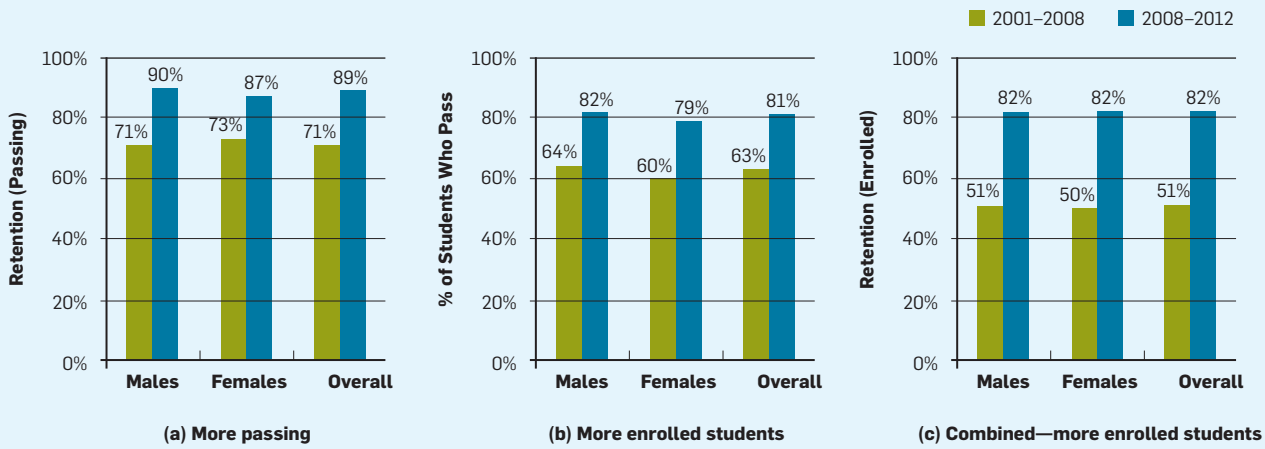
Pioneering work by Laurie Williams showed it could. Students using pair programming in an upper-division CS course produced higher-quality programs and learned the material faster. The idea is that students learn from the collaboration and the discussion, so the effort of coordinating work in a pair leads to better learning.

Charlie McDowell, Linda Werner, and their collaborators took this one step further. At the University of California at Santa Cruz (UCSC) they changed two sections of CS1 to use pair programming and left two sections with the usual solo work on programming assignments.[3] The researchers followed the students for one year after the first quarter course. They found that more students passed in the pairing sections (72%) versus the solo sections (63%), students were more likely to continue on into the next course (85% versus 67%), and they were more likely to have declared a CS major one year later (57% versus 34%). (Note: Many first-year students at UCSC are undeclared or have only a "proposed" major.)

Looking only at the final exam scores of the students that worked in pairs and

PHOTOGRAPH BY DIANA ARVAYO. COURTESY OF NEARSOFT INC.

Results of a course combining pair programming, peer instruction, and media computation over four years. (a) One-year retention for majors who pass introductory computing. (b) Passing rates for initially enrolled students. (c) One-year retention of initially enrolled students.



(a) More passing

(b) More enrolled students

(c) Combined—more enrolled students

those that worked alone, there was no significant difference. It is important to note that significantly more students in the pairing section persisted to the end and took the final. This resulted in a higher percentage of students passing the course in the pairing sections. It also refutes the claim that weak students fail to learn the material because their partner does all of the work.

Media Computation

Georgia Tech requires all students to take a course in computer science, including students in Liberal Arts, Architecture, and Business majors. During the first four years of this requirement, the overall pass rate was 78%, which is quite reasonable. The pass rate for students in Liberal Arts, Architecture, and Business, however, was less than 50% on average.

Guzdial and his colleagues created a new course just for students in Liberal Arts, Architecture, and Business programs. For these students, computing is more about *communication* than *calculation*. Students in these programs most often use the computer in order to communicate with digital media.

Media Computation was an approach to CS1 that explained how digital media are manipulated. Students learned about loops by changing all the pixels in a picture to compute a negative image, or all the samples in a sound in order to decrease the volume. Students learned about conditionals by removing red eye in the image without changing any other colors, or changing only part of a sound.

What was most exciting about Media Computation was that our assignments were defined in terms of *computation*, but the choice of what media to use in the assignments was up to the students. Students produced beautiful and creative works of art—in their CS1 class.

The result on retention was pretty dramatic.[2] The pass rate for students in those majors went from below 50% in the former class to 85% in the Media Computation class. The research evidence in the computing education community suggests it is not just media. Giving students a *context* in which to apply and understand computing makes it more relevant, and makes the students more successful.

We believe each of these approaches addresses a failing of traditional introductory computing courses.

Peer Instruction

Many of us have had the experience lecturing to a class where we have explained a key concept with brilliant clarity. We turn to the class, ask if there are any questions, and we hear... crickets. One-half of the class is looking at phones or laptops, one-fourth looks utterly confused and scared, and another one-fourth looks bored. No one asks anything, you think "they've got it" and move on. After the exam you discover: they didn't get it.

Peer Instruction, originally developed by Eric Mazur for teaching physics, seeks to remedy this problem by engaging students in the learning process. Peer Instruction modifies the standard "lecture" to revolve around 3–5 questions per lecture. For each of these questions, students follow the PI process: individually think about the problem and answer (often using clickers); discuss the question in groups; then answer again (using a clicker). Lastly, the instructor leads a class-wide discussion on the question and dynamically adjusts their explanation based on student performance.

Peer Instruction in physics has consistently shown twofold improvements in student performance on concept inventory exams versus standard lecture in large multi-institutional stud-

ies. Although Peer Instruction is new to computing, computer science education research has shown that students value Peer Instruction in upper and lower division classes, instructors value Peer Instruction, students learn from peer discussion, students in Peer Instruction classes experience a 61% reduction in failure rates, and students in Peer Instruction classes outperform standard lecture by 5% on identical final exams. (All references can be found at http://www.peerinstruction4cs.org/latest-research/.)

Combining All Three at UCSD

At this year's SIGCSE Symposium, Porter and Simon[5] reported on how all three of these approaches were combined in an introductory programming course at University of California at San Diego (UCSD). They started tracking students in 2001, and in 2008, created a new quarter-long course that combined pair programming, peer instruction, and media computation. After running the new course for four years, the results were remarkable. Not only were more students *who passed the class* retained into the Sophomore year (section a in the figure), but because more students also passed among those who initially enrolled (section b in the figure) the combined effect had a dramatic impact on retention of students enrolling in CS1 (section c in the figure).

Why Did More Students Succeed?

What is going on in these three reform efforts that cause this large change in retention? We believe each of these approaches addresses a failing of traditional introductory computing courses. We hear an often-repeated set of complaints about computer science education:

▸ *Computer science is asocial. Students see it being about sitting in the corner and hacking for hours on end, and that's just not attractive.* Pair programming shows students that being in computer science is about an intense social experience, and that learning and performance in computer science is made better by working with others.

▸ *Computer science is tedious, boring, and irrelevant.* Media Computation shows students that computer science is a creative endeavor, where the output can be beautiful. At UCSD and Skidmore College, these classes have begun hosting campuswide art shows to showcase student work,[4] a far cry from being asocial and irrelevant.

▸ *Computer science classes are competitive, with students focused on their individual grade.* Peer instruction shows students that computer science lectures are about collaborating to learn and working together as a team—starting preparation for effective work in software development teams.

There is a natural response to these kinds of efforts: that we just made these courses "easier" or "dumbed them down." The data we present about *greater* success rates into the second year, after changing only a single course, suggests students are at least as well prepared after implementing these reforms. As long as the students are achieving *desired course outcomes*, we *should* aim to make the class easier. There is no great virtue in a difficult course that flunks out students. These results demonstrate that research-based practices can make a course "easier," with higher pass rates and higher long-term retention, while still achieving desired learning outcomes. C

References

1. Bennedsen, J. and Caspersen, M.E. Failure rates in introductory programming. *SIGCSE Bull. 39*, 2 (2007), 32–36.
2. Guzdial, M. and Elliott Tew, A. Imagineering inauthentic legitimate peripheral participation: An instructional design approach for motivating computing education. In *Proceedings of the Second International Workshop on Computing Education Research*. (2006).
3. McDowell, C., Werner, L., Bullock, H.E., and Fernald, J. Pair programming improves student retention, confidence, and program quality. *Commun. ACM 49*, 8 (Aug. 2006), 90–95; DOI: 10.1145/1145287.1145293.
4. Porter, L. and Simon, B. Fostering creativity in CS1 by hosting a computer science art show. *ACM Inroads* (Mar. 2013).
5. Porter, L. and Simon, B. Retaining nearly one-third more majors with a trio of instructional best practices in CS1. In *Proceedings of the 44th Special Interest Group on Computer Science Education Technical Symposium* (2013).

Leo Porter (lporter1@skidmore.edu) is an assistant professor of computer science at Skidmore College, Saratoga Springs, NY.

Mark Guzdial (guzdial@cc.gatech.edu) is a professor in the College of Computing at Georgia Institute of Technology in Atlanta, GA.

Charlie McDowell (charlie@soe.ucsc.edu) is a professor of computer science and Associate Dean for Undergraduate Affairs in the Basking School of Engineering at the University of California Santa Cruz.

Beth Simon (bsimon@cs.ucsd.edu) is a lecturer in computer science and engineering and the director of the Center for Teaching Development at the University of California, San Diego.