

# Panel: Cooperative Learning—Beyond Pair Programming and Team Projects

Edward F. Gehringer  
North Carolina State University  
Depts. of Computer Science & ECE  
Raleigh, NC 27695-7256  
+1 919-515-2066  
efg@ncsu.edu

Katherine Deibel  
University of Washington  
Dept. of Comp. Sci. & Engineering  
Seattle, WA 98195  
+1 206 616-7046  
deibel@cs.washington.edu

Keith J. Whittington  
Rochester Institute of Technology  
Information Technology Dept.  
Rochester, NY 14623-5608  
+1 585 475-5363  
kjlw@it.rit.edu

John Hamer  
University of Auckland  
Department of Computer Science  
Auckland, New Zealand  
+64 9 3737 599x88758  
J.Hamer@cs.auckland.ac.nz

## SUMMARY

In traditional college teaching, most class time is spent with the instructor lecturing and students passively listening and taking notes. Homework is done individually, and collaboration is penalized as cheating. Instructor-centric teaching methods have repeatedly been found inferior to *cooperative learning*, in which students work in teams on problems and projects that foster interdependence while maintaining individual accountability. In recent years, the CS education community has enthusiastically embraced two collaborative-learning practices: pair programming and team projects. But why stop there? Classroom time can be devoted to group exercises, and homework assignments can be arranged so that each student plays a role in educating other members of the class. Cooperative learning has been shown to increase retention and boost the performance of at-risk students. This panel will present cooperative-learning exercises that can be used in any class, without special hardware or proprietary software.

## Categories and Subject Descriptors

K.3.2 [Computers and Education]: Computer and information science education, Curriculum. K.3.1 [Computer Uses in Education]: Collaborative learning.

## General Terms

Measurement, Documentation.

## Keywords

Cooperative learning, active learning, at-risk students.

## 1. ED GEHRINGER

I will begin with a survey of cooperative-learning practices that have been used in computer-science classes. The most basic are “think-pair-share” exercises, where students think about a problem, discuss it with their neighbors, and then share it with

the rest of the class. Another current is for students to give each other feedback on their work, e.g., by annotating each other’s research papers, by having a team of students do a code review of other students’ programs, or by writing tests for other students’ code. These strategies have much in common with agile software-development methodologies. Bringing competition into the picture always helps motivate students, e.g., having final projects compete against each other (e.g., a prey/predator game), or playing a Jeopardy-like game to review for an exam.

I will discuss my own work in having students develop resources for a soon-to-be-published textbook on object-oriented design [1]. During the semester, each student is required to do three cooperative exercises: to make up an example illustrating a concept covered in the text, to improve a section of prose, and to write an end-of-chapter problem. These are peer-reviewed in groups and revised, with the best of them making the “playoffs,” where they compete against submissions from other groups in the class. The overall winners are transmitted to the author for possible inclusion in the text.

## 2. KATE DEIBEL

One desirable outcome for any data structures course is for students to be able to discuss the pros and cons of using a particular data structure. Similarly, when given a problem scenario, students should be able to weigh these tradeoffs and decide on using a particular data structure. As a graduate teaching assistant for an undergraduate data structures course, I have my students gain experience with these tasks by working in small groups on open-ended programming scenarios [2]. Being ungraded and of a short duration, these activities have a low pedagogical cost yet can provide a valuable exercise in both critical and creative thinking [3].

For each scenario, groups are required to reach consensus about the data structure they will use. To achieve this, they must actively discuss the benefits and tradeoffs of different solutions proposed by the group. This requires students to make critical

observations and choices and to be able to justify them. Thus, the emphasis is more on the process of choosing a solution than on the solution itself.

Moreover, as the saying goes, two heads are better than one. By pooling their creative energies, insights, and ideas, the students often propose innovative solutions. During one class, a group of students collectively reinvented the skip-list (which none of them had ever heard of before) to solve a sorting/searching problem in a large linked list. As each member played role in designing that data structure, the same solution would have been unlikely if the students had worked independently.

I will discuss issues related to implementing these group work activities, including how to create open-ended problems, encouraging students to participate, and moderating how groups present their ideas. I will also relate both my and my students' experiences of these activities.

### 3. KEITH WHITTINGTON

After teaching introductory programming courses for several years in a standard lecture mode, I found that many students tended to be disengaged and unmotivated. I felt I needed to change the way that I taught these courses. I decided to incorporate cooperative learning because of its proven benefits for creating genuine communities within classrooms and promoting deep learning [4]. This technique also gives students opportunity to talk, listen, read, write, reflect, and apply what they are learning. It increases interest and participation in class, gives students more time to think, and increases student self-esteem and confidence [4].

I found that very little of the published material about cooperative learning related to teaching programming. So I developed my own activities specifically designed for introductory programming courses targeted for at-risk student who struggled in the first programming course. Students who received low Bs and C's were strongly advised to take these courses. This resulted in significant increases in student retention, grades, and self-confidence [5]. As a result of these courses, retention through the programming sequence increased by 14%, the number of A, B, and C grades increased by 15%, decreased their feelings of intimidation from fellow students by over 40%, and 89% of the students taking these courses rated it favorably. The active learning sections have consistently had an 8% D, F, W (withdrawal) rate as compared to 28% D, F, W rate in the traditional sections (which used the same exams and assignments.) The courses were so successful that the president of RIT described my work as an exemplary example for increasing student retention and satisfaction in his white paper entitled "Becoming a Category-of-One University". I also received an NSF CCLI grant this year based on these materials and their preliminary successes.

Over the years, I have developed a sizable inventory of cooperative learning activities [6]. These include first day activities, group questions, code analysis, role playing, think-pair-share, group coding, debugging activities, code design, and several others. I also come to class with a "bag of tricks" with various items that I use to help get the entire class motivated, engaged, and willing to participate.

### 4. JOHN HAMER

Inspired by a talk by Betty Collis [7], I finally decided to step into the deep end of cooperative learning. Out went the traditional assignments, with their ticky-tacky replication. In came students preparing course resources: notes, posters, slide presentations, lab solutions, software visualizations, quiz and exam questions, etc. Lectures became class meetings, usually (but not always) chaired by me. Each had an agenda, with items added by students. A minute-taker was elected, and the minutes were usually posted onto the Web by the following day (if not sooner). A "Wiki" Web was used to coordinate the material being produced. In laboratory sessions, students would write their reports onto the Wiki, and then critique other students' reports. Peer assessment served not only to manage the diverse marking workload, but ensured each student was familiar with resources other than their own. The learning objectives stretched well beyond the narrow confines of the technical topics, yet nothing needed to be dropped from the course to make room.

Student reactions varied across the full spectrum. Some were textbook Perry [8] dualism: "lecturers are there to teach, students to learn!" Others embraced the course with great enthusiasm. My favorite reaction was that of a student who wrote a careful critique of the course, in an attempt to show how the approach was inappropriate. Unfortunately, he ended up concluding the course matched closely the skills he wanted as a graduate!

I will discuss what I think worked, and why, and the parts I will need to manage better next year. Technology has a key part to play in collaborative learning, and so I will also recommend some of the tools that make things possible.

### 5. REFERENCES

- [1] Skrien, Dale, *An Introduction to Object-Oriented Design and Design Patterns Using Java*, McGraw Hill, 2006 (forthcoming).
- [2] Deibel, K., "Team formation methods for increasing interaction during in-class group work," *ITiCSE '05*: pp. 291–295, 2005. ACM Press.
- [3] L. Springer, M. E. Stanne, and S. S. Donovan. Effects of small-group learning on undergraduates in science, mathematics, engineering, and technology. *Review of Educational Research*, 69(1):21–51, 1999.
- [4] Millis, B. J. and P. G. J. Cottell, *Cooperative Learning for Higher Education Faculty*, Westport, CT, Oryx Press, 1998.
- [5] Whittington, K. J., D. P. Bills, et al., "Implementation of Alternative Pacing in an Introductory Programming Sequence," *Proc. 4<sup>th</sup> Conf. on Information Technology Curriculum*, Lafayette, Indiana, ACM Press, 2003.
- [6] Whittington, K. J., "Infusing Active Learning Into Introductory Programming Courses." *The Journal of Computing Sciences in Colleges* 19(5): 249–259, 2004.
- [7] Collis, B., "The contributing student: A blend of pedagogy and technology," keynote presentation, EDUCAUSE Australasia Conference 2005 (6 Apr.), Auckland, New Zealand.
- [8] Perry, William G., Jr., *Forms of Intellectual and Ethical Development in the College Years: A Scheme* (New York: Holt, Rinehart, and Winston, 1970), or see <http://www.cs.buffalo.edu/~rapaport/perry.positions.html>