AC 2007-2328: ACTIVE AND COLLABORATIVE LEARNING STRATEGIES FOR TEACHING COMPUTING

Edward Gehringer, North Carolina State University

Edward F. Gehringer is an associate professor in the Department of Computer Science, North Carolina State University. His research interests include hardware and software support for memory management, architectures for security, object technology, and educational software for collaborative learning.

Active and Collaborative Learning Strategies for Teaching Computing

Edward F. Gehringer
North Carolina State University
efg@ncsu.edu

Abstract

This paper is a survey of dozens of active and collaborative learning strategies that have been used in teaching computing. 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. Teams may work together in class to solve problems, with the instructor providing written comments. 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. Many such activities can be carried out online in a lab, or using laptops. A "scavenger hunt" gets students to work in pairs to surf the Web for answers to questions posed by the instructor. In another kind of exercise, students can be assigned to come up with new examples, or exercises, for the text, and then submit to an online peer-review system, where their work is reviewed by others, and the best work selected to be presented to future classes. Another strategy is to have students prepare resources to share with the class. They post these on a wiki, and the lectures become class meetings with an agenda posted on the wiki. The instructor moderates the meeting, and a student takes the minutes, and posts them on the Web. Peer assessment is used for all contributions. The paper concludes with a list of resources that include many more active and cooperative learning exercises.

1. Introduction

In bygone days, the "sage on the stage" was seen as the consummate teacher. Authoritative and entertaining, his words were eagerly listened to by students, and dutifully copied into spiral notebooks. But today, the competition is tougher. Students grow up with interactive games, watch video on their cellphones, and surf the Web from their laptops during class. To be sure, spellbinding lecturers still exist, but most of us would not count ourselves among them. We can still use class time to deliver an abridged oral rendition of the textbook, but the majority of students will see it as a waste of time. We can do better.

Active and collaborative learning strategies (ACL) are a good alternative. They hold the students' interest and facilitate learning. Leading scholars in cognitive science and educational methodologies such as Patricia Cross[5] identify active learning as an underlying principle of good practice in teaching.

The Perry model[25] is another way to view student development. The Perry model characterizes students' intellectual development in terms of their view of knowledge, the roles of instructors and students, the role of peers in the learning process, how evaluation of work should occur, and their intellectual capabilities. The model consists of nine stages that characterize the student in these dimensions, where the later stages represent greater intellectual development. One of the

characteristics of the later stages of development that distinguishes them from the earlier stages is a view toward peers as a legitimate source of knowledge or learning.

Hundreds of studies have been conducted, and meta-analyses[14, 16] show that cooperation is markedly more effective than individual learning. It has proven effective in maximizing learning for racial minorities[23] and women.[17] In the past decade, this work has become prominent enough that the reader is probably somewhat familiar with it. Many of us would like to use active and cooperative approaches in our own classes, but lack enough insight about how to apply the principles to the subject matter that we teach. This paper is intended to serve as a road map to the extensive literature that exists on active and collaborative learning strategies for computer-science courses.

It is not the goal of this work to tell you *how* to go about integrating ACL into your classes. For that, I would recommend the series by Jeffrey McConnell[19, 20, 21, 22] in *Inroads* over the past two years. Rather, this paper attempts to survey how others have used ACL in computer science and computer engineering classes, in the hope that you might run across some practices that you can adopt. Moreover, by collecting a large number of techniques in one place, I hope to sufficiently familiarize readers with the practices that they will have no difficulty thinking up exercises of their own.

In *Active Learning: Cooperation in the College Classroom*,[15] Johnson, Johnson, and Smith list five essentials for cooperative learning (for a good summary, see Fellers[9]):

- Positive interdependence. Students must realize that their contribution is essential to the success of the group. The exercise should be designed so that a "looking out for number one" approach will cause the whole group to fail.

- Face-to-face interaction. Students have similar backgrounds—at least more similar to each other than they are to the faculty. This gives them an advantage when it comes to teaching their peers, since they will have a better idea of what they might be having trouble with. Empathy also plays an important role.

- Individual accountability. While the exercises are carried out in a group, it is also important that each individual's contribution be tracked. "Social loafers," who ride along on the contributions of others, must be made to realize that their laziness will hurt the team. In some exercises, one might give the entire team a score that depends partially on the score of the lowest-scoring member.

- Social skills. Among these are communication, decision-making, and conflict resolution.

- Group process. Students need to learn how to work together as a team. The instructor needs to make sure they understand their responsibilities.

Now that we have discussed active and cooperative learning and its role, let us turn our attention to specific techniques.

2. Interactive techniques for discussion

Many active and cooperative learning techniques take the form of focused discussion. For the purposes of this paper, we will divide them into three categories: techniques that can be used to teach any topic, those that are particularly focused on computer programming but not on particular concepts in programming, and those that are useful for teaching a single concept in computer science.

a. Interactive techniques for covering any topic

i. When and how to pose questions. Questions can be used effectively to make students think. I am among the instructors who, when lecturing, use Word rather than PowerPoint because it is much easier to make changes on the fly. I will sprinkle my notes liberally with questions, and leave space to type in the answers when they are given by students. McConnell[20] says that questions should be spaced at intervals of 10–15 minutes, and require students to think about the material just presented, not just repeat answers. Instead of taking answers from the floor, you can have the students write down answers before they give them; this gives more students a chance to answer, as it prevents the quickest students from dominating the interaction. Be careful not to move on too quickly. If students know you'll provide the answer, they may not put forth the mental effort to think of the answer themselves. Wait a long time, so that students realize that class will not continue until they provide an answer.

ii. Think-pair-share. This is one of the most basic ACL techniques. Pose a question; have students think about it for a short period of time. Then, ask them to discuss it with a neighbor. After they have a few moments to do this, have one or more of the groups share their answer with the class.[20] Timmerman and Lindgard[32] found that mostly-introverted CS students were unprepared for impromptu discussions of the controversial questions they encountered in societal-issue (cf. ethics in computing) classes. They had better success by telling the students the questions in advance.

iii. Short list. List six to ten keywords or topics on the board, identifying each with a letter or a number. Take volunteers, or call on students to select one of the terms and give a brief verbal explanation. Then cross out this word and go on to the next one. Interject supporting or clarifying remarks, or allow other students to make supporting comments.[26]

iv. Online scavenger hunt. Pose a set of questions about the material covered during the lecture. Have students pair up to surf the Web for answers. (In my experience, it is not a problem to get half the students in the class to bring laptops to class; of course this assumes that the classroom has wireless access.) Each team submits a document that can be checked by the instructor and shared with the class. Ludi[18] reports, however, that this was not as well received as several other active-learning techniques.

v. Collaborative answers. Randomly split the class into groups of three. Pose a question, e.g., "Why are attributes typically declared as private, while accessors and mutators are public?" One student gives an answer. The next student improves it. The third student improves the answer given by the second. Then one group is called to the front to share their answer with the class; the class and instructor critique it.[36]

vi. Use transparencies. Some answers can be presented orally, but others (e.g., diagrams, code snippets, tracing through an algorithm) require visual display. In this case, transparencies and pens can be distributed to the groups, which can write their answer on them, and quickly present it to the class when called upon.[20] (A higher-tech way of accomplishing the same thing is to use DyKnow [http://www.dynkow.com], which allows to call up student submissions for display.) Allow students to correct any mistakes;[20] that's less threatening than having the instructor do it.

vii. Use colored cards. To get a quick indication of how well a class (especially a large class) is understanding some topic, distribute a set of colored cards (e.g., red, yellow, blue) to each student. Pose a multiple-choice question, and have students hold up a particular color to indicate a particular answer. (A higher-tech way of accomplishing this is through the use of clickers.[6, 13]) An extension of this technique has students discuss their answer with someone who is holding up a card of a different color. This technique has been mentioned by McConnell,[19] who credits Gerald Feldman for implementing a technique published by Weimer.[35] Bergin[4] also cites it.

viii. Discuss articles submitted by students. The next two techniques have initiating the topics for discussion. Timmerman and Lindgard[32] reported that discussing articles brought in by students worked well in a class on computer security. Students felt like they were experts on the topics they provided.

ix. On the hot seat. Students write down questions they would like to have answered, or questions they think their peers would want answered. In the next class session, the instructor passes out the questions. Four students are selected to be "on the hot seat" and answer the questions. The instructor can interject or clarify.[26] This, though, could be a high-risk activity, since some students may get flustered when peppered with questions.

x. Final learning check. At the end of class, it is often a good idea to have students explain key ideas in their own words.[8] When I do this, I provide a Web form that can be filled out by students who have brought laptops to class; that way I get more feedback, and it is easier to review the next time I teach the class. Of course, I also take paper submissions from students without laptops.

b. Interactive techniques for teaching programming

i. Pair programming. Pair programming is now well enough known that it hardly needs an introduction. However, several authors, including Whittington[36] admonish instructors to make sure that driver and navigator switch roles regularly, since a passive student may just let a more assertive student do all the programming.

ii. Collaborative code. Randomly split the class into groups of three. Have each group write a small snippet of code to do the same thing; for example, they might write it on a transparency. Then the group displays its solution to have the class and the instructor critique it.[36] Whittington gives an example of writing code where *HourlyEmployee* and *SalariedEmployee* inherit from *Employee* and recursively from *Person*.

iii. Collaborative code revision. Assign students one or two small programming problems to solve before class. Students bring their solutions and break into groups. The groups compare the individual solutions and arrive at a group solution. All students will be familiar with the problem, so they will be in a good position to understand others' solutions.[12]

iv. Scaffolding. Have students finish a partially written program. Give them the comments and have them write the code. Or give them the code and have them write the comments.[12, 34]

v. Error hunt. Tape printouts of about ten different code segments around the classroom. Number them from one to ten. Divide the students into groups of three. Have the groups visit each of the printouts and identify the error.[26] Pigford says this works very well with SQL, since the code is quite compact.

vi. Mystery program readings. Divide the class into groups of three or four. Each group is given a different "mystery program" that contains parallel programming constructs, and a set of questions about the code to uncover how it actually works. Using transparencies, the groups display their code snippet and the answers to their mystery questions.[28] In addition to asking what the code does, it is also possible to ask the class to predict what will happen if a certain change is made, or to make assertions about the code, or state invariants.

vii. Collaborative tracing. A group is given a code segment and asked to trace through it. Each group member is given a different responsibility. For example, if the code illustrates parameter passing by value and by reference, a number of methods might be provided, each with four parameters. Some of the parameters are passed by value and some by reference. Each member is asked to keep track of a different parameter. This keeps each group member involved.[12] Astrachan[1] gives another way of illustrating parameter passing: Use a Frisbee, and write a value on the Frisbee and toss it to represent pass-by-value. Tie a string to it and pass the string to represent pass-by-reference.

viii. Sequential programming assignments. Students often don't appreciate the problems of writing large programs because all they ever get to write are short assignments. To attack this, programming assignments can build on each other throughout the term. This requires them to give more thought to program design, and possibly to revise their old design to accommodate a new extension.[36]

c. Interactive techniques for particular topics

i. Interfaces. This exercise helps students to learn how to define interfaces. Each group is given an animal to define, and told to list attributes and behaviors of their animal. Each student needs to list two of each. Then the students present their lists, and the class attempts to find common features and arrange the animals in an inheritance hierarchy.[36]

ii. Sorting. Each group of students is given a deck of playing cards and asked to find an algorithm to sort them. First, the individuals are asked to sort them, while being timed with a stopwatch. Then the group is asked to devise an algorithm that will finish faster than the individual sorting algorithms. The groups' times are reported, and they discuss differences in their algorithms.[19]

iii. Finding second-largest item in a list. This is a good algorithm for beginners to solve in a group, because it may take some discussion to understand what is being asked for. Solving the problem helps students to deepen their understanding of programming concepts.[20]

iv. Designers and coders. Groups are divided into designers and coders. Designers write pseudocode to describe a solution. The coders examine the pseudocode and decide whether it is sufficiently detailed to be a basis for a solution, and if there are any bugs in it. If there are any problems, they return it to the designers for another attempt. If it is OK, they proceed to code it in Java.[2]

3. Toys and games

The techniques we have mentioned so far involve discussion among student groups, or between student groups and the instructor. There are many other approaches competitions among the students, or having them use a prop or act out some exercise.

a. Games

i. Dyads. There are many simple games described on the www.thiagi.com Web site (choose "Free Resources" and then "Training Games"). One such game is Dyads. Participants write out a question on the material covered during the class, then pair up with another member of the class. Each one of the pair asks the other his/her question. If the answer is correct, the questioner writes his/her initials on the other person's card. If it is incorrect, (s)he explains the correct answer. After finishing the first question, the members each find another partner and attempt to answer the new partner's question. The winner is the member with the highest score.

ii. Row competitions. Among the simplest of games is just to pass out index cards to the class members, and have them answer a set of review questions. Each student's answers are checked by a neighboring student. Then the cards are passed to one side of the classroom, and the person at the end of each row adds the correct answers from their row. The winning row is the row with the most correct answers.

iii. Review Jeopardy. Another good way to review for a test is to mimic the TV game show Jeopardy. There are at least two variations to this game. One variation has the students writing the questions. Students are divided into teams of three or four. They are given a set amount of time, and each team is asked to write some easy, medium, and hard questions on five different topics to be covered on the exam. After the questions are collected, the game begins. Each team can choose a category and a level of difficulty. Rules are similar to the game show, except that a team is not allowed to answer its own questions.[28]

Another variation has the questions being written in advance, usually by the instructor, and uses a game board similar to the board on the TV show. Just Google "Review Jeopardy" to find many different PowerPoint and Excel templates. Joe Bergin[3] has written a Java program for the game; it is found at http://csis.pace.edu/~bergin/distro/Jeopardy.zip. Pigford[26] notes that a game like this can also be modeled on "Who Wants to Be a Millionaire."

iv. Competition among student programs. Many good programming assignments are games

(e.g., ConnectFour, Othello, checkers). An element of competition is added by having a tournament in which the student programs play against each other.[27]

v. Prizes as motivation. Pollard and Duvall[27] report that a small dollar outlay yields a surprising amount of motivation. They buy a set of prizes from a local dollar store; students get to choose them in the order that their teams place in the competition. They divide the students into five or six teams. A student is asked a question; if (s)he answers it, her team gets the points. Otherwise, another team member can answer it for half the points. If the other member misses it, another team can "steal" the question. Prizes could, of course, be awarded for any of the other games mentioned above. Perhaps it is best to introduce an element of chance[27] so that the same students do not always win.

b. Kinesthetic learning activities

Many "acting-out" activities can be used in teaching any topic; these are termed *kinesthetic learning activities* (KLAs). A large collection of KLAs is available at http://ws.cs.ubc.ca/~kla.

i. Simulating an ALU. Have students represent values loaded into registers, and have them move around when these values are operated on by the ALU, written to memory, etc. This activity has been making the rounds for at least thirty years. A more elaborate version is described by Powers.[29] Here, a series of questions is posed, which leads the students to understand why ALUs, buses, etc. are needed. Then the students proceed to act them out.

ii. Program execution. A variation is to have a single student play the role of several components at the same time; this makes a more realistic simulation possible. For example, one group member might be made the "variable manager," and keep track of all variables by writing on the board. Another member might read the program, statement by statement. A third member could be given the responsibility for doing I/O.[2]

iii. Linked lists. Students can represent nodes and pointers in a linked list.[19] Similarly, I have had students insert themselves in a binary search tree by alphabetical ordering of their names.

iv. Human mystery interface. McConnell[19] describes an activity due to Wolfman. One student acts the part of a robotic toy with a poorly defined interface. Another student attempts to figure out how to use the interface by trying things. The rest of the class observes them and uses their observations as a basis for designing a better interface.

v. Swapping numbers. Have students act out the need for a temporary location when swapping two values. Ask two students to come to the front of the room. Hand each an object, and then have them put one hand behind their back and exchange the two objects without dropping them. Clearly, this will be a real challenge. Then have another student come up, also with one hand behind his back, and help the first two students swap the objects.[20]

vi. Song parodies. Not quite a KLA, but one that has students acting nonetheless, is to parody popular songs to illustrate some concept being covered in class. Pollard and Duvall[27] report that some of their students have gone so far as to make music videos.

c. Props

Instead of students playing all the roles in illustrating a particular concept, it is of course possible to use inanimate objects.

i. Beads for linked lists. There is a certain kind of beads for babies that can be linked and unlinked to demonstrate linked lists. The "nodes" on these lists can be sorted by color[1], GV 00]. However, Grissom and Van Gorp report that only 42% of their students said this was helpful in understanding linked lists.

ii. Magnetic letters for strings. Assuming that your whiteboard is metallic, magnetic letters can be move around to illustrate string functions.[27]

iii. Play Doh and memory allocation. Play Doh and cookie cutters can be used to illustrate memory allocation.[27]

iv. Algorithm visualization. Pollard and Duvall[27] helped students to prove that any $2^n \times 2^n$ checkerboard with one square removed can be totally covered by L-shaped pieces. Students had difficulty proving this until, one semester, they were supplied with a set of L-shaped cutouts. After this, almost all groups were able to complete the proof.

4. Structuring classes

Active-learning classes are often structured much differently from traditional lectures. One scheme is described by Parrish et al.[24]

- Class begins with 10–12 minutes of motivation ("why study this?").
- A 6–8 minute exercise is used to allow students to identify potential applications of the topic.
- A 10–12 minute lecture is used to present the basic material that will allow the students to utilize the concept.
- For 15–20 minutes, the students do team exercises using the concept.
- A 10–12 minute "what if" session allows the students to begin applying what they have learned. For example, they may look at previous programs they have written and see how the new concept would have helped simplify them.
- A 10–12 minute summary by the instructor recaps what has happened during this session.

Hamer[11] reports that he lets students set the agenda, and gives them the responsibility for producing lecture slides, lab experiments, quiz and exam questions. Class meetings become, well, regular meetings, with one student elected to take the "minutes" and post them on the Web.

A common school of thought holds that lecturing is fine, but after every ten minutes or so, an active-learning exercise should be given. This helps students' attention from starting to fade, and keeps them alert for the entire class.

Now, obviously ACL activities take time away from lecturing, so the instructor will not be able to cover as much material in class. A good antidote is to make sure students come prepared for class. One way to accomplish this is to have them take an online quiz over the day's material prior to the class;[19] this also assures that they come prepared to participate in the activities of the day.

An alternative suggested by Trytten[33] is to administer the quiz during class. Typically the quiz has 6–8 questions. Students first take the quiz individually; this score counts for 40% of their grade on the quiz. Then the whole team takes the quiz as a group, for another 40% of their grade. Then the team members evaluate the each other's contribution to the team effort; the scores received here constitute the remaining 20% of their grade.

5. To probe further

A tremendous amount has been written on active and collaborative learning in general. Many of the techniques have demonstrated positive, measurable effects on student learning. Dozens of authors report on how these techniques can be applied to teach computing. This survey has only been able to scratch the surface, but I am hopeful that it has given the reader a good overview of the techniques that are available. Reading individual papers will convey good ideas, a few at a time. To peruse techniques in larger batches, become familiar with the following resources.

- For an overview of considerations in applying ACL to computing, and how to go about integrating it into your courses, see Jeffrey McConnell's series in *Inroads*.[19, 20, 21, 22]

- For a large set of games that can be played in class, see http://www.thiagi.com/games.html.

- For programming puzzles that can be turned into games, Pollard and Duvall[27] recommend the "Head First" series of books.[10, 30]

- A large set of kinesthetic learning activities can be found at http://ws.cs.ubc.ca/~kla.

- The Pedagogical Patterns Project, http://www.pedagogicalpatterns.org, is devoted to documenting teaching methods in the same way that design patterns are documented. Not all of the patterns feature active learning, but a good many do.

- A site called "Classroom Assessment Techniques," http://www.siue.edu/~deder/assess/catmain.html, includes a number of self-assessment techniques that double as active learning exercises.

Now, if you are interested in applying ACL in your own classes, you should have a good idea of how to begin. Your students will appreciate it, and according to many reports, it will pay off for you too, in the form of higher scores on course evaluation.

Bibliography

[1] Astrachan, O. 1998. Concrete teaching: hooks and props as instructional technology. In *Proceedings of the 6th Annual Conference on the Teaching of Computing and the 3rd Annual Conference on integrating Technology into Computer Science Education: Changing the Delivery of Computer Science Education* (Dublin City Univ., Ireland, August 18 - 21, 1998). ITiCSE '98. ACM Press, New York, NY, 21-24.

[2] Beck, L. L., Chizhik, A. W., and McElroy, A. C. 2005. Cooperative learning techniques in CS1: design and experimental evaluation. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education* (St. Louis, Missouri, USA, February 23 - 27, 2005). SIGCSE '05. ACM Press, New York, NY, 470-474.

[3] Bergin, J. 2005. Academic jeopardy. In *Proceedings of the 10th Annual SIGCSE Conference on innovation and Technology in Computer Science Education* (Caparica, Portugal, June 27 - 29, 2005). ITiCSE '05. ACM Press, New York, NY, 389-389.

[4] Bergin, J. 2006. Active learning and feedback patterns. In *Proceedings of PloP 2006: Pattern Languages of Programs*, to be published by Springer-Verlag.

[5] Cross, K. Patricia. 1998. What do we know about students' learning and how do we know it? Presented at the Annual Meeting of the American Association for Higher Education.

[6] Demetry, C. 2005. Use of educational technology to transform the 50-minute lecture: Is student response dependent on learning style? *Proc. 2005 ASEE Annual Conference*.

[7] Dietrich, S. W. and Urban, S. D. 1996. Database theory in practice: learning from cooperative group projects. In *Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education* (Philadelphia, Pennsylvania, United States, February 15 - 17, 1996). K. J. Klee, Ed. SIGCSE '96. ACM Press, New York, NY, 112-116.

[8] Eckstein, J., Bergin J., Sharp, H. 2002. "Explain It Yourself," Feedback Patterns, Proceedings of the 7th European Conference on Pattern Languages of Programs, 2002, UVK Universitätsverlag Konstanz GmbH, Konstanz, http://csis.pace.edu/~bergin/patterns/FeedbackPatterns.html

[9] Fellers, J. W. 1996. Teaching teamwork: exploring the use of cooperative learning teams in information systems education. *SIGMIS Database* 27, 2 (Apr. 1996), 44-60.

[10] Freeman, E., Freeman, E., Bates, B. and Sierra, K. *Head First Design Patterns*. O'Reilly, 2004.

[11] Gehringer, E. F., Deibel, K., Hamer, J., and Whittington, K. J. 2006. cooperative learning: beyond pair programming and team projects. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education* (Houston, Texas, USA, March 03 - 05, 2006). SIGCSE '06. ACM Press, New York, NY, 458-459.

[12] Grissom, S. and Van Gorp, M. J. 2000. A practical approach to integrating active and collaborative learning into the introductory computer science curriculum. In *Proceedings of the Seventh Annual Consortium on Computing in Small Colleges Midwestern Conference* (Valparaiso, Indiana, United States). Consortium for Computing Sciences in Colleges. Consortium for Computing Sciences in Colleges, 95-100.

[13] Jenkins, M. and Goo, E. 2005. Concept-based instruction and personal response systems (PRS) as an assessment method for introductory materials science and engineering, *Proc. 2005 ASEE Annual Conference.*

[14] Johnson, D. W., & Johnson, R. T., *Cooperation and competition: Theory and research*. Edina, MA: Interaction Book, 1989.

[15] Johnson, D.W., Johnson, R.T. and Smith, K., *Active Learning: Cooperation in the College Classroom*, Interaction Book Company, Edina, MN, 1991

[16] Johnson, D. W., Maruyama, G., Johnson, R. T., Nelson, D., and Skon, L., "Effect of cooperative, competitive and individualistic goal structures on achievement: A meta-analysis," *Psychological Bulletin* 89, pp. 47-62.

[17] Kuh, George D.; Pace, C. Robert; and Vesper, Nick, "The Development of Process Indicators To Estimate Student Gains Associated with Good Practices in Undergraduate Education," *Research in Higher Education* 38:4, pp. 435–54, August 1997.

[18] Ludi, S. 2005. Active-learning activities that introduce students to software engineering fundamentals. In *Proceedings of the 10th Annual SIGCSE Conference on innovation and Technology in Computer Science Education* (Caparica, Portugal, June 27 - 29, 2005). ITiCSE '05. ACM Press, New York, NY, 128-132.

[19] McConnell, J. J. 2005. Active and cooperative learning: tips and tricks (part I). *SIGCSE Bull.* 37, 2 (Jun. 2005), 27-30.

[20] McConnell, J. J. 2005. Active and cooperative learning: more tips and tricks (part II). *SIGCSE Bull.* 37, 4 (Dec. 2005), 34-38.

[21] McConnell, J. J. 2006. Active and cooperative learning: further tips and tricks (part 3). *SIGCSE Bull.* 38, 2 (Jun. 2006), 24-28.

[22] McConnell, J. J. 2006. Active and cooperative learning: final tips and tricks (part IV). In *Working Group Reports on ITiCSE on innovation and Technology in Computer Science Education* (Bologna, Italy, June 26 - 28, 2006). ITiCSE-WGR '06. ACM Press, New York, NY, 25-28. Also in *Inroads—SIGCSE Bull.* 38, 4 (Dec. 2006).

[23] Milem, Jeffrey F., "Increasing Diversity Benefits: How Campus Climate and Teaching Methods Affect Student Outcomes, " ERIC Number ED456202, in Orfield, Gary, Ed., *Diversity Challenged: Evidence on the Impact of Affirmative Action*. Cambridge, Harvard Education Publishing Group, 2001, pp. 233–249.

[24] Parrish, A., Cordes, D., Lester, C., and Moore, D. 1996. Active learning and process assessment: two experiments in an Ada-based software engineering course. In *Proceedings of the Conference on Tri-Ada '96: Disciplined Software Development with Ada* (Philadelphia, Pennsylvania, United States, December 03 - 07, 1996). S. Carlson, Ed. TRI-Ada '96. ACM Press, New York, NY, 157-161.

[25] Perry, William G., *Forms of Ethical and Intellectual Development in the College Years*. San Francisco, CA, Jossey-Bass, 1999.

[26] Pigford, D. V. 2001. Designing and implementing active learning in the computer science curriculum: an interactive tutorial. *J. Comput. Small Coll.* 17, 2 (Dec. 2001), 199-204.

[27] Pollard, S. and Duvall, R. C. 2006. Everything I needed to know about teaching I learned in kindergarten: bringing elementary education techniques to undergraduate computer science classes. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education* (Houston, Texas, USA, March 03 - 05, 2006). SIGCSE '06. ACM Press, New York, NY, 224-228.

[28] Pollock, L. and Jochen, M. 2001. Making parallel programming accessible to inexperienced programmers through cooperative learning. In *Proceedings of the Thirty-Second SIGCSE Technical Symposium on Computer Science Education* (Charlotte, North Carolina, United States). SIGCSE '01. ACM Press, New York, NY, 224-228.

[29] Powers, K. D. 2004. Teaching computer architecture in introductory computing: *why?* and *how?*. In *Proceedings of the Sixth Conference on Australasian Computing Education - Volume 30* (Dunedin, New Zealand). R. Lister and A. Young, Eds. ACM International Conference Proceeding Series, vol. 57. Australian Computer Society, Darlinghurst, Australia, 255-260.

[30] Sierra, K. and Bates, B. *Head First Java*. O'Reilly, 2005.

[31] Soh, L. 2006. Implementing the jigsaw model in CS1 closed labs. In *Proceedings of the 11th Annual SIGCSE Conference on innovation and Technology in Computer Science Education* (Bologna, Italy, June 26 - 28, 2006). ITICSE '06. ACM Press, New York, NY, 163-167.

[32] Timmerman, B. and Lindgard, R. 2003. Assessment of active learning with upper division computer science students. In *Proceedings of the 33rd Annual Frontiers in Education Conference* (Boulder, CO), ASEE/IEEE, Nov. 5–8, 2003.

[33] Trytten, D. 1999. Progressing from Small Group Work to Cooperative Learning: A Case Study from Computer Science, *Frontiers in Education* 1999, available through IEEE Xplore.

[34] Walker, H. M. 1997. Collaborative learning: a case study for CS1 at Grinnell College and Austin. In *Proceedings of the Twenty-Eighth SIGCSE Technical Symposium on Computer Science Education* (San Jose, California, United States, February 27 - March 01, 1997). J. E. Miller, Ed. SIGCSE '97. ACM Press, New York, NY, 209-213.

[35] Weimer, M. (ed). *Teaching Large Classes Well*. Jossey-Bass Publishers, San Francisco, CA, 1987.

[36] Whittington, K. J. 2004. Infusing active learning into introductory programming courses. *J. Comput. Small Coll.* 19, 5 (May. 2004), 249-259.