Cooperative Learning Instructional Methods for CS1: Design, Implementation, and Evaluation

LELAND BECK and ALEXANDER CHIZHIK, San Diego State University

Cooperative learning is a well-known instructional technique that has been applied with a wide variety of subject matter and a broad spectrum of populations. This article briefly reviews the principles of cooperative learning, and describes how these principles were incorporated into a comprehensive set of cooperative learning activities for a CS1 course. In each activity, specific roles are assigned to group members in order to highlight important concepts and to enhance the overall functioning of the group. The group processing is followed by a whole-class debriefing led by the instructor, which works in tandem with the group activity to help students improve their understanding of the material. The effectiveness of these cooperative learning activities was assessed in a series of educational research studies which spanned three academic years and included two different instructors. The results of these studies show statistically significant benefits from the cooperative learning approach, both overall and for a broad range of subgroups of students. The article concludes with suggestions for faculty members who may want to use these cooperative learning activities in the classroom, or to develop their own activities along similar lines.

Categories and Subject Descriptors: K.3.2 [**Computers and Education**]: Computer and Information Systems Education—*Computer science education, curriculum*

General Terms: Experimentation, Human Factors, Measurement

Additional Key Words and Phrases: Cooperative learning, CS1, Classroom management, pedagogy, group formation

ACM Reference Format:
Beck, L. and Chizhik, A. 2013. Cooperative learning instructional methods for CS1: Design, implementation, and evaluation. ACM Trans. Comput. Educ. 13, 3, Article 10 (August 2013), 21 pages.
DOI: http://dx.doi.org/10.1145/2492686

1. INTRODUCTION

This article reports on the design, development, and evaluation of cooperative learning materials for an introductory programming course (CS1). Students using these materials work in small groups on problems that are designed to focus attention on key concepts of programming and problem solving. Group members are assigned specific roles in order to highlight important concepts and to enhance the overall functioning of the group.

As the groups work, the instructor notes problem-solving strategies employed by the students and synthesizes these strategies into a debriefing session. During this debriefing, students are motivated (because of the group experience) to understand

This work is supported by the National Science Foundation under grant DUE-0442121.
Authors' addresses: L. Beck (corresponding author), Department of Computer Science, San Diego State University, San Diego, CA; email: beck@mail.sdsu.edu; A. Chizhik, School of Teacher Education, San Diego State University, San Diego, CA.

DOI: http://dx.doi.org/10.1145/2492686

what they did correctly and what misconceptions they had in their solution strategies. Thus, the debriefing works in tandem with the cooperative learning tasks to help students improve their understanding and their problem-solving abilities.

The next section of this article gives a brief introduction to the principles and educational theory on which cooperative learning is based. This is followed by a discussion of previous uses of cooperative learning in the teaching of scientific and technical topics, highlighting the ways in which our approach differs from these previous efforts. The remaining sections contain more detailed descriptions of the design of our instructional materials, the methodology used in our educational research studies, and the results obtained. Finally, we present conclusions and practical suggestions for readers who may want to implement this approach in their own classes.

Some of the preliminary work in this project has already been described in Beck and Chizhik [2008] and Beck et al. [2005]. However, all of the research data and analyses reported in this article are new.

2. COOPERATIVE LEARNING PRINCIPLES

Conceptual learning is the primary goal behind the use of collaborative group techniques. Research in the field of education shows that collaborative group work can facilitate learning. The instructor plays a key role in this process, by preparing students for quality interactions, structuring tasks to involve all group members, forming groups to maximize high-level discourse among group members, prompting groups to engage in quality interactions during group work, and structuring debriefing sessions to link group work to learning objectives [Webb 2009]. Importantly, each of the aforementioned variables affects learning to the extent that it promotes or hinders high-level verbal interaction during group work as well as during the subsequent debriefing with the instructor [Nussbaum 2008]. In this introduction, we explain why high-level verbal interaction is of particular importance to learning in collaborative groups.

Collaborative groups of peers facilitate learning because students

> "speak to one another on a level that they can easily understand ... they speak directly to one another ... they take the feedback of another [student] seriously and are strongly motivated to reconcile contradictions between themselves and other [students] ... informational communications between [students] often are less emotionally threatening than corrective advice from an [instructor]" [Damon 1984, page 332].

In other words, collaborative groups can embody an environment in which students feel comfortable expressing their thoughts through cooperation or conflict, thereby engaging in high-level verbal interactions as they discuss pertinent subject matter with peers [Bargh and Schul 1980; Chizhik 1998, Chizhik et al. 2009; Webb 1991]. High-level verbal interactions on computer science tasks, such as the ones used in this study, may involve interchanges such as giving explanations that link solutions to significant aspects of problems, connecting prior knowledge to new information, or placing new knowledge into practice [Veenman et al. 2005]. Moreover, as students talk with each other, their ideas regarding relevant concepts of a given group task can rise to new cognitive levels, especially during cognitive conflict in groups [Chizhik et al. 2009; Nussbaum 2008; Troyer and Youngreen 2009].

As students use each others' comments to mediate their own cognition, the collaborative process can turn into a coconstruction of ideas that the students may have not been able to achieve without social interaction [Roschelle 1996; Rogoff 1990; Vygotsky 1962]. Activity theory [Cole and Hatano 2007] suggests that skills are constructed in social interations within cultural activities that they support. Collaborative learning, therefore, provides a social space to contextualize and situate skills within a community of practice [Wenger et al. 2002]. Moreover, Vygotsky [1978] established a theory of learning that suggests that higher psychological functioning (including computer programming skills) evolves in the construction of joint social activities, prior to developing into skills that can be applied to independent problem solving. These social spaces that he called "zones of proximal development" are critical for the development of complex skills. Collaborative learning harnesses social construction processes to support learners on their way to independent problem solving.

An important effect of students' attempting to solve problems is an innate need to finish solving the problem or to find out if their solution is correct. Such cognitive perturbation is a necessity in motivation to learn [Harel and Sowder 2005], creating a fertile environment for the *debriefing*, a part of collaborative learning that is just as important as the group work itself.

The debriefing focuses on the learning objectives of the lesson and is led by the instructor with the whole class participating. During the debriefing, the instructor chooses particular groups to briefly share their problem-solving strategies as well as any disagreements that they may have had. As they do so, the instructor helps the whole class to evaluate plausible problem-solving strategies, and points out effective and efficient strategies used by the groups while also highlighting less effective strategies. Any disagreement or cognitive conflict in groups is a positive and important part of inquiry that establishes an impetus for motivation to learn more through scientific inquisition [Chizhik et al. 2009; Nussbaum 2008; Osborne 2010; Troyer and Youngreen 2009]. As a part of the debriefing, the instructor may include a mini-lecture to introduce a problem-solving strategy that was not employed by any of the groups. In contrast to traditional lecture, students are primed to connect all effective strategies noted by the instructor in the debriefing discussion to their own problem-solving experience during group work. Students' motivation to complete the problem-solving process that they began in their groups leads to high-level verbal interactions in the debriefing, thereby contributing to cognitive restructuring of understandings toward learning objectives [O'Donnell 2006].

3. PREVIOUS RESEARCH INVOLVING COOPERATIVE LEARNING

This section briefly reviews the existing educational research base related to the work reported in this article, and identifies the ways in which the current project differs from previously published work.

Cooperative learning techniques have been applied with a wide variety of subject matter and a broad spectrum of populations. In general, cooperative learning involves students working together as part of a collaborative effort to understand material or complete a task [Sharan 1990, 1994; Slavin 1995]. Accumulated evidence suggests that cooperative learning results in higher student achievement, more positive attitudes toward the subject, improved student retention, and a variety of other benefits [Johnson et al. 1991; Sharan 1994; Slavin 1995]. It appears that the cooperative learning approach may be especially beneficial for women and members of underrepresented minority groups [Nelson 1996; Sandler et al. 1996; Slavin 1995; Williams et al. 2007; Yerion and Rinehart 1995]. One possible explanation for this is that cooperative learning changes the atmosphere of the classroom. When groups work on tasks that can be solved in multiple ways, there are more opportunities for all students to contribute to their group's problem solving. This can lead to more equitable participation by minority students [Alexander et al. 2009; Chizhik 2001; Chizhik et al. 2003]. In addition, the problem-solving process becomes more supportive and less competitive. For some women and minority students (and, indeed, some members of the male majority), this may be a very positive change.

3.1. Cooperative Learning in STEM Disciplines

A number of studies have found that effective cooperative learning experiences can reduce the rate of attrition for students majoring in Science, Technology, Engineering, and Mathematics (STEM). This seems to be especially true for females and students from ethnic minority groups [Seymour and Hewitt 1997].

One STEM education initiative that deserves special mention is the Process-Oriented Guided Inquiry Learning (POGIL) project [Moog and Spencer 2008]. Students in a POGIL classroom work together in small groups on specially designed activities, with the instructor acting as a facilitator. Studies at a number of institutions have shown that the POGIL approach can lead to improved student learning and lower attrition [Hanson and Wolfskill 2000; Hinde and Kovac 2001; Lewis and Lewis 2005].

Our work has many similarities to POGIL. As in POGIL, the instructor is a leader who establishes the group norms and supports the students in learning their process-oriented roles as group members. One significant difference is in the emphasis that we place on whole-class debriefing activities following a period of group work. A typical POGIL classroom activity consists of about 5% introductory remarks from the instructor, 90% group work, and 5% closure. In POGIL, the primary pedagogical tool is for the instructor to guide the quality of interaction within each group by asking high-quality questions. While there is some evaluation in the wrap-up, the primary focus at the conclusion of the lesson is the instructor's helping the groups evaluate the quality of their interactions [Hanson 2006].

In contrast, our activities typically include about 5% introductory remarks to orient the students to the group tasks, 55% group work, and 40% debriefing, which focuses on the learning objectives of the lesson and is led by the instructor with the whole class participating. Experience and observation in the classroom suggest that this kind of whole-class discussion is an important part of the learning process [National Research Council 2005].

A number of educators have studied the use of cooperative learning techniques in computer science courses. Some of these studies are described in Chase and Okie [2000], Falkner and Palmer [2009], Gonzalez [2006], Joseph and Payne [2003], Troeger [1995], and Walker [1997]. In many cases, these efforts involve dividing students into small groups to work on programming problems or laboratory exercises. Typically, these problems or exercises are very similar to those that might be given as individual assignments in a traditional course. It is expected that students will learn from each other as they work cooperatively on an assignment; however, the assignments are not designed specifically to take advantage of the structure of the group. In most cases, any special roles are purely functional (such as having one group member working at the keyboard), rather than being related to the concepts being studied.

In contrast, our cooperative learning exercises assign specific roles to group members. Some of these roles are used to focus attention on fundamental concepts of computing, key elements of the language, and other important aspects of the topics being learned. Other roles are designed to support the overall functioning of the group. For example, one student may be assigned the role of reporter: keeping notes on the group problem-solving process and reporting on this process during the debriefing phase. Another student may be assigned the role of facilitator: helping the group process stay on track and helping other group members stay in their assigned roles. Roles are rotated from one exercise to the next, so that each group member performs a variety of tasks. Most theories of cooperative learning hold that this division of group responsibilities into roles significantly improves the effectiveness of the learning process [Cohen 1994; Johnson and Johnson 1998; Sharan 1994; Slavin 1995].

3.2. Pair Programming

An educational technique that has elements in common with cooperative learning is pair programming [Williams and Kessler 2002]. In this form of collaboration, two programmers work side by side at one computer. At any particular time, one member of the team (the "driver") is typing at the computer or writing down a design. The other member (the "navigator") is actively observing the work of the "driver", watching for defects, thinking of alternatives, asking questions, etc. The "driver" and "navigator" roles are switched periodically between the two team members.

Pair programming was originally popularized as part of the Extreme Programming software development methodology [Beck 2000]. Research results indicate that pair programmers produce higher-quality code in about half the time, as compared with programmers working alone [Williams 2000; Williams et al. 2002b]. Pair programming techniques have also been found effective for student programmers, leading to improved student learning and satisfaction and reduced frustration [Braught et al. 2008; Hanks 2006; Hanks et al. 2004; McDowell et al. 2002; Mentz et al. 2008; Williams et al. 2002b].

Our cooperative learning exercises use methods similar to pair programming to help students learn about the processes of programming and problem solving. Our exercises, however, lead students through a number of different levels of cooperation. In the early stages, for example, the entire group may be asked to brainstorm together in solving a problem. At a later stage, students work in pairs to solve a problem, and then compare their solutions with those developed by another pair in the group. Still later, other exercises give students the opportunity to work on problems on their own, with consultation from the other group members if needed. We believe that this incremental approach offers even more advantages than strict pair programming. At first, everyone in the group is learning how to approach a programming task. Thus it is useful to have as many different points of view as possible. As their programming and problem-solving skills develop, students progress to working in pairs. Finally they have the chance to build confidence by solving problems individually (still with support from the group).

Including more than two members in a collaborative group also allows us more flexibility in assigning concept-related and process-related roles as discussed previously. Some of the exercises involve a substantial number of roles, which would be more difficult to implement in a strict pair-programming environment.

4. DESIGN OF COOPERATIVE LEARNING MATERIALS FOR CS1

Our educational materials for CS1 consist of 29 cooperative learning experiences. These were initially developed for use with the Java language; they were subsequently translated into C++ by one of our collaborators, Professor Jeff Jackson of Duquesne University.

There are five basic types of activities:

(1) Introductory exercises used during the first few days of class to give an overview of the processes of programming and problem solving;
(2) Activities used to introduce the most fundamental concepts of Java/C++ (variables, objects, methods, etc.);
(3) Programming activities that introduce the syntax and semantics of language constructs, and give students practice in using them to perform simple tasks;
(4) Problem-solving activities that ask students to use previously learned language elements to design and implement solutions to more complex problems;
(5) Specialized testing and debugging experiences. (Testing is an important element of almost all of our cooperative learning activities. However, the exercises in this group are structured to focus special attention on key aspects of this essential skill.)

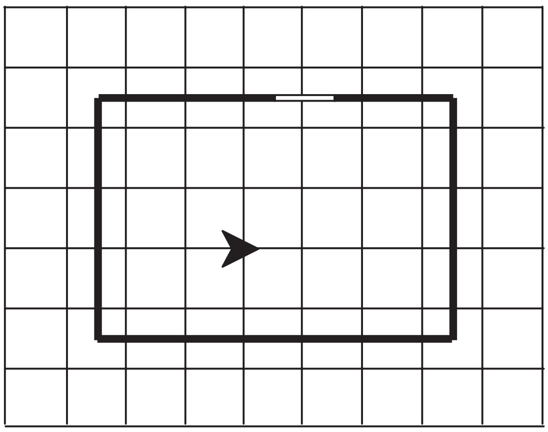

Fig. 1. Starting position for an introductory exercise.

In most of these activities, specific roles are assigned to each student within a cooperative learning group. Examples of such roles are included in the sample activities described shortly.

In addition to the student-oriented materials, our cooperative learning activities for CS1 also include a comprehensive set of instructor support materials for each exercise. These materials contain a discussion of learning objectives for the exercise, suggestions for introducing the cooperative learning activities, things to watch for during the group processing, and key points to help the instructor plan and conduct an effective debriefing session.

Two recurring problem contexts are used throughout the set of activities: a slightly modified version of the “Karel the Robot” world [Pattis 1995] and a ticket-selling application. The Karel world is intuitive and provides an easy way to introduce important programming concepts early in the course. The ticket-selling application starts as an extremely simple system, and is expanded and made more complex as students learn more advanced programming and problem-solving techniques.

These two recurring contexts provide a familiar environment in which students can practice using newly introduced language concepts and problem-solving techniques. They also provide opportunities for students to recognize similarities between problems, and motivation for the use of more advanced language features. After students have applied newly learned material in a familiar context, the set of activities leads them to apply these same new ideas in other environments.

The remainder of this section briefly describes some sample exercises to illustrate the variety and scope of these cooperative learning activities.

4.1. An Introductory Exercise

The first exercise we describe is used very early in the course, typically in the first week. This problem is set in the Karel the Robot world. Karel starts somewhere inside a closed rectangular room that has one “window” (a glass section of wall), as illustrated in Figure 1. The objective is for Karel to find the window and stop beside it. Students are given a description of an algorithm for doing this; because they do not know any programming languages yet, the algorithm is described in graphical form (similar to a flowchart). According to this algorithm, Karel first moves forward from its starting position until it reaches a wall. Then it turns left and continues along the wall, searching for the window. If it reaches a corner of the room, it turns left and continues searching

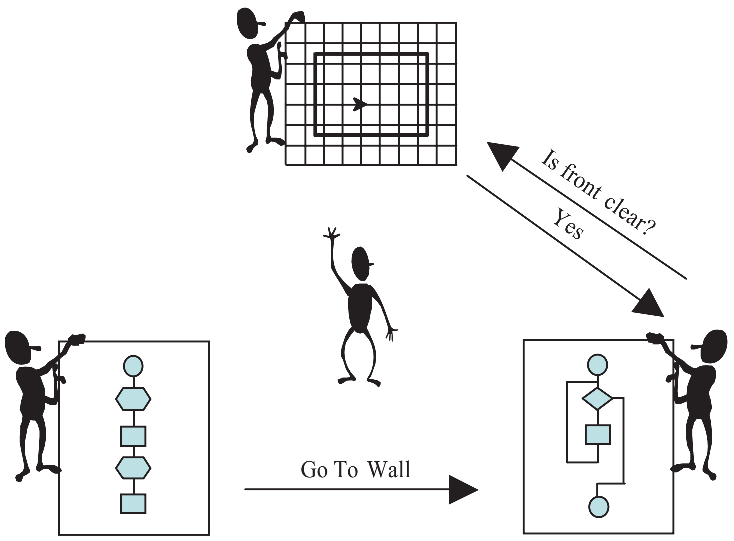

Fig. 2. A group of students working on the introductory exercise.

along the next wall. Eventually it reaches a position where the window is on its right and stops.

Figure 2 illustrates a group of students working together to execute this algorithm. The two students shown at the bottom of the figure have different parts of the algorithm description. They pass control between them as specified in the algorithm. They also pass messages to, and receive replies from, a student who is responsible for simulating the operation of the robot (shown at the top of the figure). A fourth student is serving as facilitator, helping to guide the process and making sure that group members stay in their assigned roles. (For example, a student who is executing part of the algorithm is not allowed to look at the position of the robot to see whether or not it can move forward. Instead, he or she must ask the person who is simulating the robot for this information.)

The algorithm the students are given works correctly for the starting situation shown in Figure 1. After students have successfully executed the algorithm for this case, they are told that there are other situations in which it does *not* work as expected. They are challenged to work together to find some of these situations, and to think about how the algorithm would need to be modified to fix the problems.

One error situation that students typically find is illustrated in Figure 3(a). In this case, the window is located in a corner of the room. Following the algorithm as given, Karel will move forward to the wall and then follow the wall until it reaches the corner as shown. However, the algorithm now causes Karel to turn left and step forward before it checks to see whether it is beside the window. In this situation, the robot will continue to search around the room indefinitely, never finding the window.

Figure 3(b) illustrates a more subtle problem. In this situation, Karel will move forward until it reaches the wall as shown. However, the algorithm will now cause Karel to turn left before it starts checking for the window. For this starting situation, the robot will eventually find the window and stop beside it. However, it will make an extra circuit completely around the room before it does so, probably not the behavior that we would prefer.

This exercise introduces many important concepts, all in the first day or two of class. The robot is a natural and intuitive example of an object: it has state and behavior, and

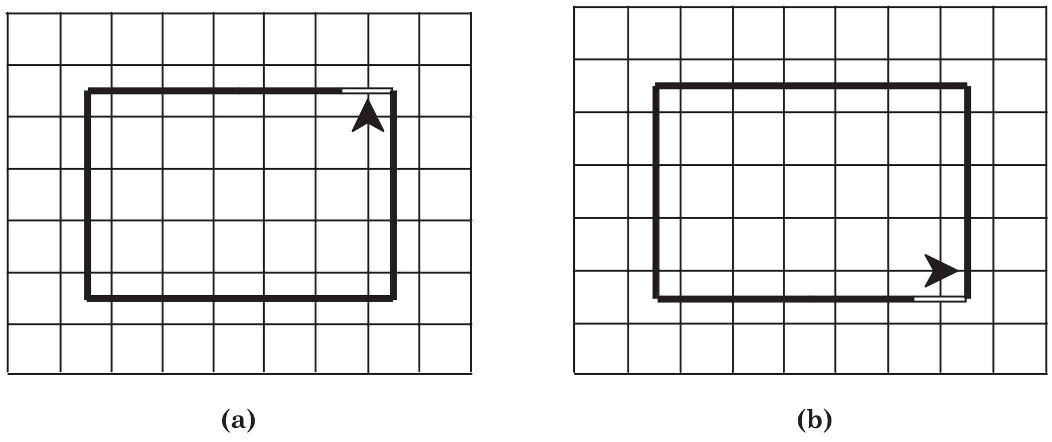

Fig. 3. Program error situations frequently detected by students.

the other parts of the program communicate with it by sending messages (essentially invoking instance methods) and receiving replies. The students who are executing different parts of the algorithm pass control among themselves (essentially invoking static methods). As students execute their parts of the algorithm, they are also gaining experience with sequential program execution, decision making, and looping.

Perhaps even more important, the students are beginning to learn higher-level skills that will be very valuable to them as they study programming. They see that a program can work correctly in some situations, but fail in others. They also begin to develop some insight into how to look for problem situations. For example, most groups notice that window positions in the middle of a wall usually lead to similar results. Problems tend to occur in "boundary" situations, such as the window being placed in a corner of the room. This observation can help guide them in testing and debugging when they later encounter other boundary situations in programs.

This exercise also leads to a useful preview of Java syntax. After students have worked with the algorithm expressed in graphical form, they are shown the same algorithm written in Java. Because students are already familiar with how the program works, this helps them to understand what Java statements like "if" and "while" are saying (even though they do not yet know how to write them). This also provides an opportunity for the instructor to introduce terms like "object" and "method" and to point out the syntax for invoking instance methods, for example, statements like karel.move() and karel.turnLeft(). This gives students a preview of things they will study later. It also helps them realize that computer programs can be logical and understandable, allowing them to start the course with a feeling of confidence.

4.2. Exercises Focusing on Basic Concepts of Java

In another early exercise, students work together in groups to simulate the execution of a simple Java program as illustrated in Figure 4. One group member (the variable manager, shown at bottom right) is assigned to keep track of names and contents of variables, using pencil and paper. A second group member (the program reader, shown at bottom left) reads a simple Java program that contains variable declarations and assignment statements, one line at a time, and issues the appropriate instructions to the variable manager.

Other group members are responsible for performing simple I/O operations, when instructed to do so by the program reader. One student acts as facilitator, as described

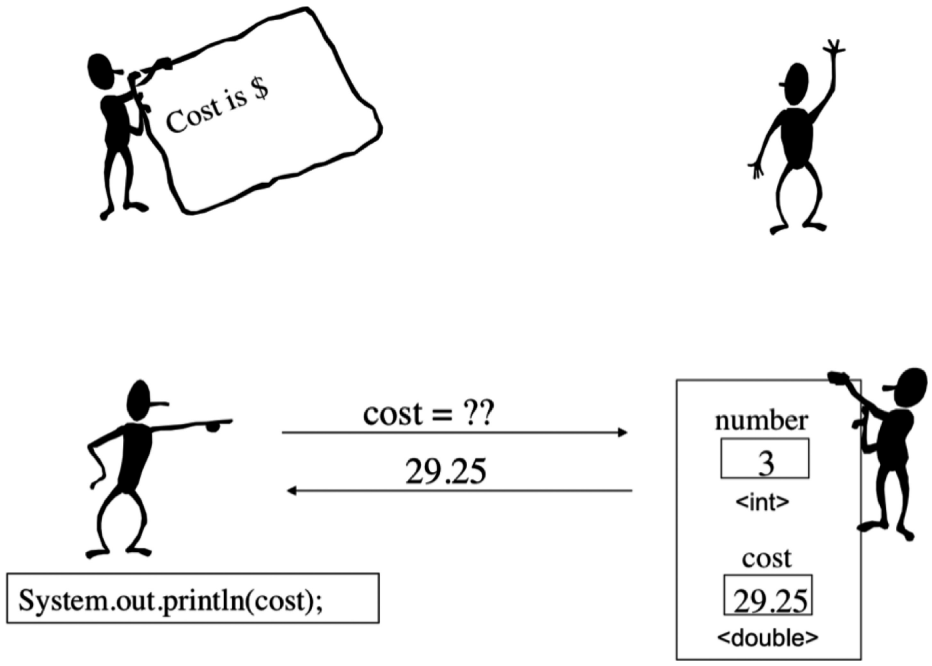

Fig. 4. A group of students working on a different type of exercise.

earlier, monitoring the process and helping the other group members stay in their assigned roles.

The division of roles in this exercise is intended to reflect the underlying process of sequential program execution and the storage and use of variables, thus helping the students to develop their own mental models of the way programs are executed by a computer. A later exercise uses roles in a similar way to develop insight into the implementation and use of objects in a Java program. In that exercise, when an object is created by the program, a student is selected to implement that particular object. Each student keeps track of the values of instance variables and executes constructors and instance methods (from the class definition) as required. Students who are implementing different parts of the program pass messages to one another to simulate the process of method invocation. Variables that contain references to objects are manipulated within the program in order to highlight the differences between these variables and the objects themselves.

4.3. Testing and Debugging Exercises

Many of our CS1 exercises are designed to focus special attention on important aspects of programming and problem solving. For example, one early testing and debugging experience presents students with short problem descriptions and segments of Java code that claim to solve these problems. The students are asked to work as a group to determine whether or not the proposed solutions are correct; if they find errors, they are asked to fix the code so that it does work correctly. The problems are constructed in a way that highlights common misconceptions that students may have as they are learning. This type of exercise helps students develop the skill of looking critically at a piece of code (their own or someone else's) and analyzing whether or not it works correctly.

Subsequent testing and debugging experiences are designed to help students enhance and refine their skills in this area. For example, one exercise presents students with the description of a ticket-pricing policy, with several different ticket categories and a system of discounts governed by a variety of conditions. Students are asked to devise a set of data that would thoroughly test a program that implements the stated policy. After they have created their test data, they are asked to use it to test several different prewritten versions of the program. All of these prewritten versions contain at least one error; many of our students have been amazed to learn that their test data actually misses several important (and plausible) errors in the code.

4.4. Problem-Solving Exercises

Other cooperative learning experiences are designed to separate and highlight important aspects of programming and problem solving. In early problem-solving exercises, one group of students is presented with a problem description and asked to write Java code to solve it. They then pass their code to another group of students for testing (perhaps exchanging code with a group that is working on a different problem). If errors are found, the code is returned to the writers with a description of the problem that was found (the test case and incorrect output, not a diagnosis of the bug in the code). The writers try to figure out how to fix the problem and send a revised version for further testing. This process repeats until the solution is determined to be correct. It is easy for the person who originally wrote a piece of code to miss seeing an error in it. Separating the code writing and testing operations in this way makes it more likely that errors will be found (serving as learning opportunities).

Later exercises use a similar approach to focus on the use of pseudocode. When an individual student is asked to write pseudocode as the first step in solving a problem, he or she may be tempted to skip that step, or may write something that does not clearly describe the solution of the problem. To avoid this, some of our exercises treat the pseudocode as a separate item. One group of students writes pseudocode to describe a solution to a given problem. The pseudocode is then passed to another group of students for critical examination and review, with a cycle of testing and correction as described before.

5. METHODOLOGY

Our evaluation of the cooperative learning approach, and of our educational materials, consisted of two related studies. Study A included seven different sections of our CS1 course over a period of three semesters. This phase of the research was a between-instructor design, with one instructor using the cooperative learning approach and the other using a more traditional lecture-oriented approach. This was followed by Study B, which included all of our sections of CS1 for the next three semesters. This phase was a within-instructor design, examining the results obtained by an instructor who changed from a lecture-oriented approach to the cooperative learning approach. The methodologies for these two studies are described in the following sections.

5.1. Study A

During each semester of Study A we had at least one cooperative learning section and at least one comparison section. The cooperative learning and comparison sections were of similar sizes, typically between 30 and 70 students. The instructors for the cooperative learning and comparison sections had several prior semesters of experience in teaching CS1.

We took care to avoid introducing bias into the teaching of the cooperative learning and comparison sections. Neither of the instructors had participated in the development of the cooperative learning materials, and neither of them was involved in the project in any other way, except for their teaching duties. Neither of the authors of this article had any direct contact with the students in the cooperative learning or comparison sections.

The comparison sections were taught in a traditional lecture-based format, with some time spent on questions and general class discussions. In the cooperative learning sections, students worked in small groups on cooperative learning exercises for 30 to 45 minutes during almost every 75-minute class meeting; the remainder of class time consisted of mini-lectures, administrative topics, and general class discussions. Students in the cooperative learning sections were assigned to cooperative learning groups during the first week of class. Most students remain in the same groups for the duration of the course. However, the instructor monitored overall group functioning, and was able to reassign groups if it appeared that this would improve group interaction. Typically, these reassignments happened after midterm exams were graded. Section 8 of this article describes some of the factors that were considered in group formation.

Students were not randomly assigned to the cooperative learning and comparison sections; they enrolled for the section that best fit their schedules. During each semester, however, the cooperative learning and comparison sections were similar on general measures of academic performance such as GPA and SAT scores. There was also a similar distribution of majors across the various sections, with about 30% of the students in each section majoring in computer science or computer engineering. A total of 232 students were involved, with 91 enrolled in cooperative learning sections and 141 enrolled in comparison sections.

During each semester, the cooperative learning and comparison sections met at different times and had different (although similar) lab assignments and midterm exams. For maximum consistency, our analysis in this research focused on the final exam for the course, which was taken by students from all sections at the same time. The instructors for the various sections collaborated in writing and refining the questions on this final exam. Care was taken to ensure that the exam did not give students from one section any special advantage. For example, none of the exam questions were similar to problems that had been discussed in one section but not in another. The students from all sections received the same sample exams, handouts, and other review materials in preparation for the final exam.

The structure of the final exam was similar during each semester of the study, consisting of three parts. The first part, which accounted for 10% of the overall score, contained multiple-choice questions that focused on simple syntactic and semantic rules. The second part, which accounted for 30% of the overall score, consisted of questions that asked students to read short pieces of Java code and trace their execution (for example, by showing the output that would be generated). The third part (60% of the overall score) contained questions that focused on the process of program writing. Some of the questions in this part asked students to write complete Java methods or short programs, according to specifications given. Other questions asked students to fill in the missing parts of a partially written program or method, or to modify an existing piece of Java code, again according to specifications given in the question.

In order to avoid any possible unconscious bias in grading, we obscured the students' names and mixed the exam papers from the different sections together before scoring them. The scoring was done by a single individual (one of the authors of this article), not by either of the instructors involved. In order to improve uniformity of scoring, we developed a rubric for assigning credit to answers on free-response questions on the exam. Scores in the rubric were defined according to key elements of each question, for example, whether an appropriate control structure was selected or whether a method header was correctly constructed.

5.2. Study B

Results from Study A showed clear benefits from the cooperative learning approach. (These results are discussed in detail in the following section.) Accordingly, we began teaching all of our sections of CS1 using this approach. We also began a second phase of the research (Study B), which was designed to study the experience of an instructor making the transition to cooperative learning.

Study B focused on the instructor who had taught CS1 in a traditional lecture format during the first three semesters of the project (Study A). We studied the achievement of his students for the next three semesters (now being taught using cooperative learning), comparing their exam scores with the scores of his students from the previous phase.

The overall methodology for this part of the research was very similar to that used previously. We used the same cooperative learning exercises, and students worked in groups on these exercises for 30 to 45 minutes during most 75-minute class periods. The instructor formed and monitored groups using the same criteria, and conducted debriefing sessions following the same guidelines. The final exams had the same overall structure, and were constructed with questions that were as similar as possible to questions that had been used during the first phase of the research. The exams were independently scored for research purposes by the same person as in the first phase, using the same rubrics.

5.3. Threats to Validity

As described earlier, we took a number of precautions to minimize threats to validity in our studies. For example, we were careful to design and administer the final exams so that neither group of students had a special advantage. We scored the exams using a uniform rubric, without knowing which group of students had written each exam paper. We did our best to separate the instructional and research processes, so that the instructors did not have a vested interest in the outcome.

However, there are some threats that we were not able to avoid. Students were self-selected into the cooperative learning and comparison groups, based on which section they chose when registering. It was not practical to randomly assign students to sections, because many of them had class or work schedules that would have created conflicts. The only way to allow random assignment would have been to schedule all of our CS1 sections at the same time. This was not considered academically feasible, because it would have prevented some students from taking the course.

We checked for possible differences in the groups of students being studied, using a t-test to compare overall GPA and SAT scores. For Study A, we compared the various sections that were being offered during each semester. We also compared the aggregated groups (including all three semesters) that were used in our analyses of results in Study A and Study B. There were no statistically significant differences in any of these comparisons; however, we cannot rule out the possibility that other factors might have influenced the results.

A more serious threat to validity arises from the use of different instructors in Study A. We do not have data to compare the performance of the instructors prior to the start of our research. It is quite possible that there were inherent differences in teaching style or instructional skills that affected the results of Study A. In fact, there are indications that point in this direction (as discussed in Section 7 later).

We might have been able to avoid this threat by having cooperative learning and comparison sections taught by the same instructor. However, this was not practical because of the instructors' other teaching commitments. Having the same instructor teach using two different styles would also have introduced a different possible source of bias.

The two-instructor threat to validity does not exist in Study B. However, it is still possible that the results of Study B were affected by characteristics of the particular

Table I. Cooperative vs Lecture, Study A (Different instructors, First 3 Semesters)

	Achievement on the Entire Final Exam				*Achievement on Program Writing Part of Exam*			
Subgroup	Coop.-Learning Sections Mean (SD), N	Comparison Sections Mean (SD), N	t	p	Coop.-Learning Sections Mean (SD), N	Comparison Sections Mean (SD), N	t	p
All Students	72.2 (18.6), 92	56.8 (19.2), 141	6.07	<.001	69.6 (31.5), 92	48.9 (33.3), 141	4.75	<.001
Computer Science Major	73.1 (20.9), 22	57.8 (21.2), 50	2.82	<.01	69.1 (35.5), 22	54.5 (34.7), 50	1.63	n.s.
Math/ Science Major	73.9 (18.3), 39	56.1 (18.7), 40	4.26	<.001	71.3 (29.7), 39	47.5 (32.4), 40	3.41	<.001
Non-Science Major	69.6 (17.6), 31	56.4 (17.8), 51	3.28	<.01	67.9 (31.7), 31	44.5 (32.4), 51	3.20	<.01
European American	77.3 (14.5), 40	58.4 (15.7), 52	5.89	<.001	76.3 (24.8), 40	45.8 (29.6), 52	5.25	<.001
Non-European American	68.4 (20.5), 52	55.9 (21.0), 89	3.44	<.001	64.5 (35.2), 52	50.7 (35.3), 89	2.25	<.05
Female	68.9 (16.0), 27	56.4 (16.4), 40	3.10	<.01	63.9 (27.7), 27	47.4 (31.3), 40	2.22	<.05
Male	73.6 (19.5), 65	57.0 (20.3), 101	5.24	<.001	72.0 (32.9), 65	49.5 (34.2), 101	4.21	<.001

instructor involved. A comparison of the results of Studies A and B (see Section 7) offers some insights into this question, as well as raising questions that could be addressed by future research.

6. RESULTS AND ANALYSIS

As previously described, the initial phase of our research (Study A) spanned three semesters. During each semester, we had at least one traditional section (taught by Instructor 2) and at least one cooperative learning section (taught by Instructor 1). In our analyses, we compared the final exam scores achieved by students in these two types of sections.

In addition to the overall exam scores, we also compared the scores on the third part of the exam (the part that focused on program writing). The questions in the first two parts of the exam dealt with facts and mechanical skills such as code tracing. In contrast, this third part was designed to test higher-level cognitive skills in program design and synthesis. Educational theory suggests that cooperative learning may be especially effective in developing these kinds of skills.

We conducted a preliminary analysis of our data to verify the assumptions for the use of parametric statistics. Levene's test for equality of variances did not show statistically significant differences for any of the comparisons (all p's > 0.1). Skewness and kurtosis were between -1.0 and $+1.0$ for all subsamples being compared. Tests of normality indicated that each sample was not significantly different from a normal distribution. Accordingly, we used multivariate ANOVA and associated t-tests for the comparisons in our analysis.

As Table I of associated t-tests shows, our analyses in Study A found that cooperative learning sections taught by Instructor 1 performed statistically significantly better on the final exam than traditional sections taught by Instructor 2. The MANOVA detected statistically significant effects of instructional section, $F(2, 589) = 7.5$, $p < .001$; this effect was statistically significant for the entire final exam, $F(1, 4854) = 11.7$, $p < .001$, partial $eta^2 = .019$, as well as for the program-writing part of the exam, $F(1, 107) = 15.1$, $p < .001$, partial $eta^2 = .025$. We detected a similar differential in performance by all subgroups, including males, females, various majors (computer science, mathematics

Table II. Cooperative vs Lecture, Study B (Same Instructor, First 3 Semesters vs Second 3 Semesters)

	Achievement on the Entire Final Exam				*Achievement on Program Writing Part of Exam*			
Subgroup	Coop.-Learning Sections Mean (SD), N	Comparison Sections Mean (SD), N	t	p	Coop.-Learning Sections Mean (SD), N	Comparison Sections Mean (SD), N	t	p
All Students	59.0 (20.0), 269	56.8 (19.2), 141	1.04	n.s.	55.2 (25.6), 269	48.9 (33.3), 141	2.14	<.05
Computer Science Major	58.6 (19.6), 97	57.8 (21.2), 50	0.23	n.s.	56.4 (24.5), 97	54.5 (34.7), 50	0.40	n.s.
Math/Science Major	63.06 (19.7), 78	56.1 (18.7), 40	1.84	n.s.	59.5 (24.6), 78	47.5 (32.4), 40	2.25	<.05
Non-Science Major	55.8 (20.2), 94	56.4 (17.8), 51	0.14	n.s.	50.4 (26.9), 94	44.5 (32.4), 51	0.89	n.s.
European American	63.4 (21.6), 89	58.4 (15.7), 52	1.47	n.s.	58.6 (27.2), 89	45.8 (29.6), 52	2.61	<.01
Non-European American	56.8 (18.8), 180	55.9 (21.0), 89	0.34	n.s.	53.5 (24.6), 180	50.7 (35.3), 89	0.77	n.s.
Female	54.4 (19.2), 74	56.4 (16.4), 40	0.56	n.s.	50.2 (23.9), 74	47.4 (31.3), 40	0.54	n.s.
Male	60.7 (20.1), 195	57.0 (20.3), 101	1.50	n.s.	57.1 (26.0), 195	49.5 (34.2), 101	2.14	<.05

and science, as well as nonscience), and different ethnic groups (European Americans and non-European Americans).

In addition, Instructor 1's cooperative learning sections in Study A performed better than Instructor 2's traditional sections on "program writing" items in the third part of the exam. Likewise, all subgroups, except for students with computer science majors, demonstrated similar differences, with cooperative learning sections performing better than traditional sections (see Table I).

In Study B, we compared the scores achieved by Instructor 2's students during the last three semesters (using cooperative learning) with scores from Instructor 2's traditional sections during the first three semesters. The MANOVA detected statistically significant effects of instructional section, $F(2, 407) = 3.9$, $p < .05$; on the entire final exam differences were not statistically significant, but were statistically significant on the program-writing part of the exam, $F(1, 30) = 4.6$, $p < .05$, partial $\text{eta}^2 = .011$. These differences were also detected in associated t-tests for a number of subgroups, including math/science majors, European Americans, and males (see Table II).

As we were scoring the exams, we noticed that students in cooperative learning sections seemed to leave far fewer questions blank, especially on the program-writing part of the exam. We believe that this is important because it indicates a difference between the two groups of students in self-efficacy (the perception of their own ability to solve programming problems). Cooperative learning has been found to improve students' self-efficacy in other disciplines [Fenci and Scheel 2005]. Studies have also found that self-efficacy is correlated with variables such as interest in the subject and persistence in scientific and technical majors [Lent et al. 1987].

We also believe that the number of questions attempted is an indicator of problem-solving potential that goes beyond the scores achieved on an exam. A student who makes an attempt to solve a problem—even if that attempt is not correct—may be able to use his or her testing and debugging skills to arrive at a correct solution. On the other hand, a student who is unable even to attempt an initial solution has very little hope of solving the problem.

To investigate this question further, we examined the proportion of students from the various sections who left at least half of the program-writing questions blank (not even attempting an answer to them). As Table III shows, substantially fewer students from the cooperative learning sections during the first three semesters fell into this no-response category, as compared with traditional lecture sections. The difference

Table III. Cooperative vs Lecture, Study A (Different instructors, First 3 Semesters), Percent of Students Leaving 50% or More Blanks on Program-Writing Items

Subgroup	Coop.-Learning Sections Percent of Students, N (Expected N)	Comparison Sections Percent of Students, N (Expected N)	Chi Square	p
All Students	5.4, 5 (11.1)	16.3, 23 (16.9)	6.23	<.05
Computer Science Major	13.6, 3 (3.4)	16.0, 8 (7.6)	0.07	n.s.
Math/Science Major	2.6, 1 (2.5)	10.0, 4 (2.5)	1.84	n.s.
Non-Science Major	3.2, 1 (4.5)	21.6, 11 (7.5)	5.19	<.05
European American	0, 0 (3.5)	15.4, 8 (4.5)	6.74	<.01
Non-European American	9.6, 5 (7.4)	16.9, 15 (12.6)	1.41	n.s.
Female	3.7, 1 (2.0)	10.0, 4 (3.0)	0.93	n.s.
Male	6.2, 4 (9.0)	18.8, 19 (14)	5.31	<.05

Table IV. Cooperative vs Lecture, Study B (Same Instructor, First 3 Semesters vs Second 3 Semesters), Percent of Students Leaving 50% or More Blanks on Program-Writing Items

Subgroup	Coop.-Learning Sections Percent of Students, N (Expected N)	Comparison Sections Percent of Students, N (Expected N)	Chi Square	p
All Students	2.6, 7 (19.7)	16.3, 23 (10.3)	25.64	<.001
Computer Science Major	3.1, 3 (7.3)	16.0, 8 (3.7)	7.94	<.01
Math/Science Major	1.3, 1 (3.3)	10.0, 4 (1.7)	4.95	<.05
Non-Science Major	3.2, 3 (9.1)	21.6, 11 (4.9)	12.80	<.001
European American	1.1, 1 (5.7)	15.4, 8 (3.3)	11.17	<.001
Non-European American	3.3, 6 (14.1)	16.9, 15 (6.9)	15.13	<.001
Female	2.7, 2 (3.9)	10.0, 4 (2.1)	2.77	n.s.
Male	2.6, 5 (15.8)	18.8, 19 (8.2)	23.58	<.001

between sections was statistically significant overall; this difference also appeared in every subgroup shown in the table, although many of the subgroup differences are not significant because of the smaller numbers involved.

Table IV shows the results of the same analysis, comparing Instructor 2's cooperative learning sections (in the last three semesters) with his lecture sections (in the first three semesters). Again, there were far fewer students in the no-response category for the cooperative learning sections. These differences were statistically significant overall, and also for all of the subgroups except females.

7. DISCUSSION AND CONCLUSIONS

In the first phase of our research (Study A), we compared sections of CS1 taught during the same semester using two different approaches: cooperative learning (taught by Instructor 1), and a traditional lecture-based approach (taught by Instructor 2). The data and analyses in Table I indicate a clear benefit for the cooperative learning approach (as implemented by Instructor 1), with statistically significant improvements in performance across a broad range of students.

For the second phase of the research (Study B), Instructor 2 changed to the cooperative learning approach. His students also improved in performance as compared with his previous lecture-based sections (Table II). The amount of improvement overall was not as great as for Instructor 1's students using cooperative learning, with most results not being significant. In fact, we found statistically significant differences between Instructor 1's cooperative learning students (in Study A) and Instructor 2's cooperative learning students (in Study B). This suggests that at least part of the improvement observed in Study A was due to the difference in instructors.

However, Table II does show statistically significant improvements for Instructor 2's students on the program-writing part of the exam, which required students to apply higher-level cognitive skills. We also note that Instructor 2's students improved significantly in the number of program-writing questions they attempted to answer (Table IV). This suggests that the cooperative learning experience helped them to develop a higher level of confidence in their own problem-solving abilities. Thus, we conclude that the cooperative learning approach was also effective for Instructor 2, although apparently to a lesser degree than for Instructor 1.

We can think of several possible explanations for this difference between instructors. Instructor 1 had taught CS1 using cooperative learning as part of a pilot study, which occurred before the research reported in this article. On the other hand, Instructor 2 was using cooperative learning for the first time during the beginning of phase 2 of our research. We provided training and consultation to both instructors when they were using the cooperative learning approach. However, this consultation was more detailed and extensive during the first phase of the research (with Instructor 1) than during the second phase (with Instructor 2). It is also possible that cooperative learning was a better match for the personality and natural teaching style of Instructor 1.

Regardless of these factors, however, it seems clear that students in both instructors' classes benefited from the cooperative learning approach. We believe that it would be useful to conduct further studies focusing on instructor-related factors that influence the outcomes obtained from cooperative learning.

One aspect of the analysis that deserves special comment is the relatively low frequency of statistically significant results for computer science majors (see Tables I, II, and III). In each case, computer science majors performed better with the cooperative learning approach. The improvement, however, was smaller than for most other subgroups, and often fell short of statistical significance. We believe that this is at least partially due to the composition of our classes. In a typical semester, fewer than one-third of our CS1 students are CS majors. These students have a special interest in programming, and perhaps also a special aptitude or experience. However, because of the diversity of the student population, the classes may not be taught at a pace that stretches these students' abilities. We would expect to see more statistically significant results for computer science majors in classes that are designed to move at a faster pace.

As previously discussed, several studies of cooperative learning have found particular benefits for students from underrepresented minority groups. However, many of our results for non-European American and female students were not statistically significant (see Tables II, III, and IV). Some of this lack of significance is probably due to the small numbers of students in these subgroups. However, we also observe that the overall improvement in average score tends to be smaller for these students than for students in the European American and male majorities.

We note that the amount of improvement for non-European American and female students was larger for Instructor 1's cooperative learning sections than it was for Instructor 2's sections. (Compare the values shown for cooperative learning groups in Tables I and II.) This may have been in part because Instructor 1 was more experienced, and perhaps more skilled, in managing the cooperative learning process. We believe that this is an issue that deserves further research.

8. SUGGESTIONS FOR IMPLEMENTATION

The cooperative learning activities described in this article are designed so that they can be used in a variety of different sequences, according to the instructor's preference. We would be glad to share all of our materials with anyone who may be interested in using or adapting them for use in the classroom. We would also gladly provide training and consultation, and help in any other way we can. Please contact one of the authors if you are interested in this possibility.

We also recognize that some instructors may prefer to develop cooperative learning materials for their own particular educational environment, perhaps following an approach similar to the one described here. Regardless of the specific materials used, there are common questions and issues that are worth considering. Students' engagement in high-level verbal interactions and establishing cognitive perturbation in small groups is dependent on their feeling comfort in expressing thoughts in a small group of peers. This section focuses on factors that influence high-level verbal participation and comfort in peer interactions.

8.1. Group Formation

Research suggests that group composition by relevant ability level has an influence on learning outcomes [Webb 1991, 2009]. For example, relevant abilities on a computer programming task, in addition to including actual programming skills, also include ability to solve logical and mathematical problems. A proxy for such abilities would be prior experience in programming as well as course grades in relevant subjects. In particular, relatively high- and low-ability students benefit when placed in a group together. The same research, however, suggests that when relatively low-, medium-, and high-ability students are placed in the same group, the relatively medium-level students do not gain as much from the interaction as do other group members. Webb [1991] found that medium-level students tend to benefit from being a part of homogeneous groups where all members are of relatively medium ability. That finding can be related to Webb's [1991] and, subsequently, Veenman et al.'s [2005] reporting that the benefits of group work come primarily from giving explanations and receiving prompt answers to questions. In heterogeneous groups the high-level students are more likely to provide explanations while the low-level students are more likely to receive prompt responses from high-level students. The medium-level students within heterogeneous groups are less likely to participate verbally because the high-level student is more likely to provide the majority of the explanations. Medium-level students, however, seem to prosper the most in homogeneous groups where they can participate verbally by grappling with issues and coconstructing solutions to complex problems.

Research suggests that groups with sole females or sole males perform worse than those that include more than one member from each gender [Sormunen-Jones et al. 2000]. There is also evidence that unequal distribution of males and females in groups may impair performance of female participants in comparison with groups that include an equal number of males and females [Webb 1984]. Similarly, research argues against forming groups that include only one member whose ethnicity is different from the other group members [Webb et al. 1997]. These findings stem from a theoretical perspective that gender and ethnicity are general status characteristics on which group members base expectations for participation and quality of contributions to the group work [Cohen et al. 2002; Webster and Rashotte 2010]. The implication of these findings for classes where there are unequal distributions of genders and ethnicities is that some groups would be homogeneous by the gender or ethnicity that is overrepresented.

8.2. Monitoring and Adjusting Group Composition

As previously noted, the quality of learning in groups is ultimately based on the level of interaction among all group members [Nussbaum 2008]. Therefore, while initial grouping should be based on the theoretical notions presented earlier, instructors should monitor group performance and interactions and adjust group composition to maximize quality participation for all students.

In determining the quality of group interactions, the first step should be to evaluate the overall level of group performance. In a small classroom, an instructor may be able to observe each group and determine the extent to which each group is discussing issues pertinent to skills necessary to solve the problem at hand. In a large classroom, however, an instructor would need to require groups, and possibly each student, to record their process of problem solving in a systematic way. These notes can then be used to evaluate the quality of each group's problem solving.

Once the instructor has determined which groups may have lower levels of quality discourse, she or he should observe those groups more carefully to determine what may be causing lower performance. Does the group not have the necessary skills to solve the problems and needs a member with relatively high ability? Is the group misreading the directions and going astray in their problem solving? Are some students not getting along to the extent that they cannot work together? Are some students being left out of the conversation? Are some students choosing to stay out of the conversation?

The intervention could be to instruct the group and help them develop group norms that will improve the level of discourse (see next section). Alternatively, it may be necessary to adjust the groups. Once you get to know your students, you should be able to place higher-ability students in groups where previously students did not have the prerequisite skills to solve problems. Likewise, you should also be able to determine which students are more patient in working with a student who was not able to get along in another group. If possible, all adjustments should be made simultaneously to avoid singling out any students. It is important to note that adjustment of group composition may decrease the performance of some groups who have lost a higher-ability student or have gained a student who has trouble getting along with others. These are opportunity costs that are associated with the difficult job of the instructor to establish and maintain the overall quality of instruction within the class.

8.3. Influence of Group Norms on High-Level Interaction

Educational researchers have investigated the types of verbal interactions that are most likely to facilitate learning in collaborative groups [Cohen 1994; Webb and Palincsar 1996]. For example, Webb [1991, 2009] reports that verbal interactions in the form of elaborated explanations facilitate learning for group members who actually give the explanations. In addition, Webb [1991] found that students who receive prompt answers to their questions tend to improve their individual performance, while students whose questions are ignored tend to show a decline in individual performance. Therefore, it is necessary for instructors to explicitly foster and nurture elaborated explanations, questioning, and prompt responses to questions as norms within collaborative groups [Veenman et al. 2005]. This can be done by instructors' directly informing the whole class as to the types of discourse that are expected while working in groups, having students participate in mock scenarios that highlight the importance of these types of group interactions, and directing individual groups toward these types of participations during actual problem-solving tasks.

As the instructor evaluates groups, even those that seem to be performing well overall, they may notice that some students "loaf" without having to do any work [Webb and Palincsar 1996]. In that situation, the "loafer" does not learn from the group project, and the student who initially does all the work may stop working because she or he feels like a "sucker" [Webb and Palincsar 1996]. The "loafer" and "sucker" effects are most common when the entire task can, in fact, be accomplished by a single group member. To address this issue of higher-level students doing the work for the low-level students, the instructor should work to establish and maintain group norms that encourage each group member to be responsible for particular roles within collaborative groups [Cohen and Lotan 1995].

When developing classroom norms, it is important for teachers to keep in mind classroom culture as well as the societal culture of students while emphasizing equity in high-level verbal interactions. For example, Lewis [1997] reports that giving power in collaborative groups to students with higher status (in the study, status was based on grade level and verbal ability) supports a classroom culture in which the opinions of unempowered students are devalued. Such classroom norms do little to deemphasize students' preexisting expectations of each other and, in effect, may strengthen those expectations. In light of this, instructors should establish and regularly rotate the role of facilitator, often seen as the most powerful role in the group [Esmonde 2009]. More importantly, however, instructors should help groups develop norms whereby students ask each other questions in order to promote higher participation of all group members and not "boss" each other around. For example, instead of facilitators commanding group members to "do your job," they or any other group member may ask a group member who is perceived as not participating what he or she thinks about a particular idea or for their way of solving an aspect of the problem. Ultimately, instructors should attempt to establish norms that promote an equitable environment in which all students have opportunities to participate in high-level verbal interactions.

ACKNOWLEDGMENTS

We gratefully acknowledge the contributions of the instructors at SDSU who participated in this research study along with their students. We would also like to acknowledge the following faculty members at other institutions who gave us valuable feedback regarding our approach and materials: Lawrence Angrave, Jay Bockelman, Darrah Chavey, Erica Eddy, Debby Keen, Bill McClung, Elsa Mentz, Neil Mittal, David Taylor, Deborah Trytten. (We apologize to anyone whose name has been inadvertently omitted.) We would especially like to thank Jeff Jackson of Duquesne University, who converted our cooperative learning materials into C++ versions, and has graciously agreed to share the results of his efforts with others.

REFERENCES

ALEXANDER, M. G., CHIZHIK, A. W., CHIZHIK, E. W., AND GOODMAN, J. A. 2009. Lower-status participation and influence: Task structure matters. *J. Social Issues 65*, 2, 365–381.

BARGH, J. A. AND SCHUL, Y. 1980. On the cognitive benefits of teaching. *J. Educ. Psychol. 72*, 5, 593–604.

BECK, K. 2000. *Extreme Programming Explained: Embrace Change.* Addison-Wesley, Reading, MA.

BECK, L. L. AND CHIZHIK, A. W. 2008. An experimental study of cooperative learning in CS1. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'08).* ACM Press, New York, 205–209.

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 (SIGCSE'05).* ACM Press, New York, 470–474.

BRAUGHT, G., EBY, L. M., AND WAHLS, T. 2008. The effects of pair-programming on individual programming skill. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'08).* ACM Press, New York, 200–204.

CHASE, J. D. AND OKIE, E. G. 2000. Combining cooperative learning and peer instruction in introductory computer science. In *Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education (SIGCSE'00),* S. Haller, Ed., ACM Press, New York, 372–376.

CHIZHIK, A. W. 1998. Collaborative learning through high-level verbal interaction: From theory to practice. *The Clearing House 72,* 1, 58–61.

CHIZHIK, A. W. 2001. Equity and status in group collaboration: Learning through explanations depends on task characteristics. *Social Psychol. Educ. 5,* 179–200.

CHIZHIK, A. W., SHELLY, R. K., AND TROYER, L. 2009. Intragroup conflict and cooperation: An introduction. *J. Social Issues 65*, 2, 251–259.

CHIZHIK, A. W., ALEXANDER, M., CHIZHIK, E. W., AND GOODMAN, J. 2003. The rise and fall of power and prestige orders: Influence of task structure. *Social Psychol. Quart. 66*, 3, 303–317.

COHEN, E. G. 1994. Restructuring the classroom: conditions for productive small groups. *Rev. Educ. Res. 64*, 1, 1–35.

COHEN, E. G., LOTAN, R. A., AND ABRAM, P. L. 2002. Can groups learn? *Teachers College Rec. 104*, 6, 1045–1068.

COHEN, E. G. AND LOTAN, R. A. 1995. Producing equal-status interaction in the heterogeneous classroom. *Amer. Educ. Res. J. 32*, 1, 99–120.

COLE, M. AND HATANO, G. 2007. Cultural-historical activity theory: Integrating phylogeny, cultural history, and ontogenesis in cultural psychology. In *Handbook of Cultural Psychology,* S. Kitayama and D. Cohen, Eds., Guilford Press, New York, 109–135.

DAMON, W. 1984. Peer education: The untapped potential. *J. Appl. Devel. Psychol. 5*, 4, 331–343.

ESMONDE, I. 2009. Mathematics learning in groups: Analyzing equity in two cooperative activity structures. *J. Learn. Sci. 18*, 2, 247–284.

FALKNER, K. AND PALMER, E. 2009. Developing authentic problem solving skills in introductory computing classes. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education (SIGCSE'09).* ACM Press, New York, 4–8.

FENCI, H. AND SCHEEL, K. 2005. Engaging students. *J. College Sci. Teach. 35,* 1, 20–24.

GONZALEZ, G. 2006. A systematic approach to active and cooperative learning in CS1 and its effects on CS2. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'06).* ACM Press, New York, NY, 133–137.

HANKS, B. 2006. Student attitudes toward pair programming. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITICSE'06).* ACM Press, New York, 113–117.

HANKS, B., MCDOWELL, C., DRAPER, D., AND KRNJAJIC, M. 2004. Program quality with pair programming in CS1. In *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE'04).* ACM Press, New York, 176–180.

HANSON, D. M. 2006. *Instructor's Guide to Process-Oriented Guided Inquiry Learning.* Pacific Crest, Lisle, IL.

HANSON, D. M. AND WOLFSKILL, T. 2000. Process workshops—A new model for instruction. *J. Chem. Educ. 77,* 120–130.

HAREL, G. AND SOWDER, L. 2005. Advanced mathematical thinking at any age: Its nature and its development. *Math. Thinking Learn. Int. J. 7*, 1, 27–50.

HINDE, R. J. AND KOVAC, J. 2001. Student active learning methods in physical chemistry. *J. Chem. Educ. 78,* 93–99.

JOHNSON, D. W. AND JOHNSON, R. T. 1998. *Learning Together and Alone* 5th Ed. Allyn and Bacon, Boston, MA.

JOHNSON, D. W., JOHNSON, R. T., AND SMITH, K. A. 1991. *Active Learning: Cooperation in the College Classroom.* Interaction Book Company, Edina, MN.

JOSEPH, A. AND PAYNE, M. 2003. Group dynamics and collaborative group performance. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'03).* ACM Press, New York, 368–371.

LENT, R. W., BROWN, S. D., AND LARKIN, K. C. 1987. Comparison of three theoretically derived variables in predicting career and academic behavior. *J. Counsel. Psychol. 34*, 3, 293–298.

LEWIS, C. 1997. The social drama of literature discussions in a fifth/sixth-grade classroom. *Res. Teach. English 31*, 163–204.

LEWIS, S. E. AND LEWIS, J. E. 2005. Departing from lectures: An evaluation of a peer-led guided inquiry alternative. *J. Chem. Educ. 82,* 135–139.

MCDOWELL, C., WERNER, L., BULLOCK, H., AND FERNALD, J. 2002. The effects of pair-programming on performance in an introductory programming course. In *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education (SIGCSE'02).* ACM Press, New York, 38–42.

MENTZ, E., VAN DER WALT, J. L., AND GOOSEN, L. 2008. The effect of incorporating cooperative learning principles in pair programming for student teachers. *Comput. Sci. Educ. 18,* 247–260.

MOOG, R. S. AND SPENCER, J. N., Eds. 2008. *Process Oriented Guided Inquiry Learning (POGIL).* American Chemical Society, Washington, DC.

NATIONAL RESEARCH COUNCIL. 2005. *How Students Learn: History, Mathematics, and Science in the Classroom.* The National Academies Press, Washington, DC.

NELSON, C. E. 1996. Student diversity requires different approaches to college teaching, even in math and science. *Amer. Behav. Scientist 40,* 165–175.

NUSSBAUM, M. E. 2008. Collaborative discourse, argumentation, and learning: Preface and literature review. *Contemp. Educ. Psychol. 33,* 3, 345–359.

O'DONNELL, A. M. 2006. The role of peers and group learning. In *Handbook of Educational Psychology,* 2nd Ed, P. Alexander and P. Winne, Eds., Lawrence Erlbaum, Mahwah, NJ, 781–802.

OSBORNE, J. 2010. Arguing to learn in science: The role of collaborative, critical discourse. *Sci. 328*, 5977, 463–466.

PATTIS, R. E. 1995. *Karel The Robot: A Gentle Introduction to the Art of Programming* 2nd Ed., John Wiley and Sons, New York.

ROGOFF, B. 1990. *Apprenticeship in Thinking: Cognitive Development in Social Context.* Oxford University Press, New York.

ROSCHELLE, J. 1996. Learning by collaborating: Convergent conceptual change. In *CSCL: Theory and Practice of an Emerging Paradigm,* T. Koschmann, Ed., Lawrence Erlbaum, Mahwah, NJ, 209–248.

SANDLER, B. R., SILVERBERG, L. A., AND HALL, R. M. 1996. *The Chilly Classroom Climate: A Guide to Improve the Education of Women.* National Association for Women in Education, Washington, DC.

SEYMOUR, E. AND HEWITT, N. M. 1997. *Talking about Leaving: Why Undergraduates Leave the Sciences.* West View Press, Boulder, CO.

SHARAN, S., ED. 1990. *Cooperative Learning: Theory and Research.* Praeger Publishers, Westport, CT.

SHARAN, S., ED. 1994. *Handbook of Cooperative Learning Methods.* Praeger Publishers, Westport, CT.

SLAVIN, R. E. 1995, *Cooperative Learning: Theory, Research, and Practice* 2nd Ed. Allyn and Bacon, Boston, MA.

SORMUNEN-JONES, C., CHALUPA, M. R., AND CHARLES, T. A. 2000. The dynamics of gender impact on group achievement. *Delta Pi Epsilon J. 42*, 154–170.

TROEGER, D. 1995. Formal methods, design, and collaborative learning in the first computer science course. *New Directions Teach. Learn. 61*, 55–66.

TROYER, L. AND YOUNGREEN, R. 2009. Conflict and creativity in groups. *J. Social Issues 65*, 2, 409–427.

VEENMAN, S., DENESSEN, E., VAN DEN AKKER, A., AND VAN DER RIJT, J. 2005. Effects of a cooperative learning program on the elaborations of students during help seeking and help giving. *Amer. Educ. Res. J. 42*, 115–151.

VYGOTSKY, L. 1962. *Thought and Language.* MIT Press, Cambridge, MA.

VYGOTSKY, L. 1978. *Mind in Society: The Development of Higher Psychological Processes.* Harvard University Press, Cambridge, MA.

WALKER, H. M. 1997. Collaborative learning: A case study for CS1 at Grinnell College and UT-Austin. In *Proceedings of the 28th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'97).* J. E. Miller, Ed., ACM Press, New York, 209–213.

WEBB, N. M. 1984. Sex differences in interaction and achievement in cooperative small groups. *J. Educ. Psychol. 76*, 33–34.

WEBB, N. M. 1991. Task-related verbal interaction and mathematics learning in small groups. *J. Res. Math. Educ. 22,* 5, 366–89.

WEBB, N. 2009. The teacher's role in promoting collaborative dialogue in the classroom. *Brit. J. Educ. Psychol. 79,* 1, 1–28.

WEBB, N. M., BAXTER, G. P., AND THOMPSON, L. 1997. Teachers' grouping practices in fifth-grade science classrooms. *Elementary School J. 98*, 91–113.

WEBB, N. M. AND PALINCSAR, A. S. 1996. Group processes in the classroom. In *Handbook of Research in Educational Psychology,* D. Berliner and R. Calfee, Eds., Prentice-Hall, London, 841–873.

WEBSTER, M. AND RASHOTTE, L. S. 2010. Behavior, expectations, and status. *Social Forces 88,* 3, 1021–1049.

WENGER, E., MCDERMOTT, R., AND SNYDER, W. M. 2002. *Cultivating Communities of Practice.* Harvard Business School Press, Boston, MA.

WILLIAMS, L. 2000. The collaborative software process. Ph.D. dissertation, University of Utah, Salt Lake City, UT.

WILLIAMS, L. AND KESSLER, R. 2002. *Pair Programming Illuminated.* Addison-Wesley, Reading, MA.

WILLIAMS, L., KESSLER, R., CUNNINGHAM, W., AND JEFFRIES, R. 2002a. Strengthening the case for pair programming. *IEEE Softw. 19,* 19–25.

WILLIAMS, L., LAYMAN, L., SLATEN, K. M., BERENSON, S. B., AND SEAMAN, C. 2007. On the impact of a collaborative pedagogy on african american millennial students in software engineering. In *Proceedings of the 29th International Conference on Software Engineering.* IEEE Computer Society, Los Alamitos, CA, 677–687.

WILLIAMS, L., WIEBE, E., YANG, K., FERZLI, M., AND MILLER, C. 2002b. In support of pair programming in the introductory computer science course. *Comput. Sci. Educ. 12*, 197–212.

YERION, K. A. AND RINEHART, J. A. 1995. Guidelines for collaborative learning in computer science. *SIGCSE Bull. 27,* 4, 29–34.

Received March 2012; revised November 2012; accepted April 2013