Implementing CS1 With Embedded Instructional Research Design in Laboratories

Jeff Lang, Gwen C. Nugent, Ashok Samal, and Leen-Kiat Soh, *Member, IEEE*

***Abstract*—Closed laboratories are becoming an increasingly popular approach to teaching introductory computer science courses. Unlike open laboratories that tend to be an informal environment provided for students to practice their skills with attendance optional, closed laboratories are structured meeting times that support the lecture component of the course, and attendance is required. However, as observed in <AUTHOR: In abstract, references cannot be used since the abstract is sometimes accessed alone; please rewrite to make reference unnecessary—ed.>[1], "Considering the prevalence of closed laboratories and the realization that they have been in place in the Computer Science (CS) curricula for more than a decade, there is little published evidence assessing their effectiveness." This paper reports on an integrated approach to designing and implementing laboratories with embedded instructional research design. The activities reported here are parts of a departmentwide effort not only to improve student learning in computer science and computer engineering (CE) but also to improve the agility of the Computer Science and Engineering Department in adapting the curriculum to changing technologies, incorporate research, and validate the instructional strategies used. This paper presents the design and implementation of the laboratories and the results and analysis of student performance. Also described in this paper is cooperative learning in the laboratories and its impact on student learning.**

***Index Terms*—Computer science education, laboratories, cooperative learning, instructional design, introductory computer science/computer engineering (CS/CE) courses.**

I. Introduction

RAPID and continuous change in the areas of software development and information technology pose significant pressure on educational institutions in terms of educating and training the next generation of professionals. In particular, maintaining the curriculum in a computer science degree program is a challenge that requires constant attention. The Association for Computing Machinery (ACM) and the IEEE Computer Society, the two leading professional bodies in the field of computer science, have recently released guidelines outlining core topics for a computer science degree program [2]. Subsequently, the Department of Computer Science and Engineering at the University of Nebraska—Lincoln initiated a review of its own undergraduate program in computer science with the long-term goal of redesigning and reorganizing the computer science (CS) curriculum to improve the quality of instruction and student learning. After careful consideration, the department approved an innovative curriculum that has the potential to improve significantly the quality of undergraduate computer science education [3]. One of the key innovations is the application of a traditional, science-based (e.g., physics, chemistry, and biology) approach to computer science laboratories, supported by research in educational psychology and instructional design. The scope of this project included introductory CS courses (CS0, CS1, and CS2). The innovation will lead to a more rigorous curriculum for CS majors and a better understanding of fundamental CS topics for the students.

Closed laboratories are becoming an increasingly popular approach to teaching introductory computer science [1], per the recommendations by Denning *et al.* [4] and ACM's Computing Curricula 1991 [5]. Closed laboratories have multiple advantages. Students learn at the beginning of their majors to be active learners through goal-oriented problem solving in a laboratory setting [6]. Doran and Langan [7] demonstrated that laboratories promote students' cognitive activities in comprehension and application, in terms of Bloom's taxonomy [8]. One study, in fact, reported that even though the closed laboratories did not help improve retention or project completion rates in the CS1 course, a qualitative improvement in student learning was evident in the closed laboratory sections [9]. Thweatt reported a statistical design with an experimental group (closed laboratory), and a control group (open laboratories) for a CS1 course and found that students in closed laboratories consistently performed significantly better on comprehensive CS1 exams than those in open laboratories [10]. Further, exploration opportunities help first-time programmers overcome common hurdles, such as misconceptions about the nature of computers and programs [11]. Parker *et al.* found that closed laboratories demonstrate the scientific method of inquiry and teach skills in data collection and analysis [12]. The laboratory environment also facilitates cooperative learning among students [13]. Finally, laboratories tend to provide a more flexible environment that can cater to students of different backgrounds and learning styles. With all that is known regarding the benefits of closed laboratories, "there is little published evidence assessing their effectiveness" [1]. Indeed, there are many online documents on laboratory designs for CS curriculum without explicit evaluation or assessment components [14]–[17].

This paper presents a systematic approach to design, implement, assess, and evaluate closed laboratories. To demonstrate the effectiveness of the approach, the research team focused on measuring the impact of cooperative learning in laboratory

Manuscript received July 26, 2005; revised October 28, 2005. This work is supported in part with funding from National Center for Information Technology in Education.

J. Lang and G. Nugent are with the National Center of **<AUTHOR: of or for (see bio)?—ed.>**Information Technology in Education (NCITE), University of Nebraska, Lincoln, NE 68588-0230 USA.

A. Samal and L.-K. Soh are with the Department of Computer Science and Engineering, University of Nebraska, Lincoln, NE 68588-0115 USA.

Digital Object Identifier 10.1109/TE.2005.863435

0018-9359/$20.00 © 2006 IEEE

IEEE Proof

settings and analyzing the results pedagogically. Further, other qualitative measurements were provided to complement the analysis and discussions. Readers are referred to [18] for an analysis of the relationships between the laboratories and course activities in CS1, and [19] for additional results of educational research in CS1 laboratories. While this study has focused on CS courses, the educational research methods and results may be broadly applicable to other structured engineering laboratories.

In Section II, related work on laboratories for introductory CS courses and cooperative learning are presented. Subsequently, laboratory design and implementation approach is presented in Section III. **<AUTHOR: Please verify section numbers are correct—ed.>**Section IV discusses the various results and analysis, with cooperative learning as the focus of the analysis. Finally, current and future work is discussed in the Conclusion section.

II. Background

A. Related Work on Laboratories

Since 1990, several laboratory designs for introductory programming or computer literacy courses have been implemented. For example, a set of CS1 and CS2 laboratories were designed to introduce students to computing environments, such as editing, system use, file manipulation, and compilation [20]. Other approaches include laboratories designed for high-volume computer literacy [21], and declarative laboratories for discrete structures, logic, and computability [22]. Bruce *et al.* created a set of XML-based laboratory exercises for CS1, using templates to create standardized HTML laboratory exercise sets from XML documents [23]. The authors focused on the development of these XML documents from the viewpoint of standardization, customization, and simulation. An education forum on lecture and laboratory syllabus also provided for a breadth-first introductory CS course sequence posted in 1993 [24]. However, the paper, being a forum document, did not elaborate on how the laboratories should be designed in terms of sequencing, pedagogy, and instructional research. In the design of the laboratories at the University of Nebraska—Lincoln (UNL), a comprehensive pedagogical approach that included educational research design, implementation, and validation was used.

Geitz reported on a laboratory design that included four parts: concepts, applications, programming exercises, and write-up, so that the laboratories focus on programming using the concepts learned in the class [25]. The concepts section discusses the new material covered in the laboratory, and the students are expected to have read this section prior to the start of the laboratory. The applications section is a walkthrough of several programs. The programming exercises section asks the students to write complete programs or extend existing ones. Finally, the write-up section asks students questions designed to make them think about what they have learned in the laboratory. Geitz reported increases in student retention since the introduction of the new curriculum, which included the laboratories [25]. The design used in this study was similar to Geitz's design except some of the laboratories included new material not covered in the lectures, such as debugging and testing.

Lischner proposed a set of guidelines for including explorations in CS laboratories [11]. An exploration is a structured dialog with the student: the student reads a short program, answers questions about that program, makes predictions about the program's behavior, and then tests the predictions by running the program and answering follow-up questions. If a prediction is wrong, the student is asked to give a plausible explanation. Guidelines suggested by Lischner include 1) the most effective explorations are short and to the point; 2) an exploration should have a variety of questions, including some that are impossible to answer correctly; 3) explorations should direct the student toward effective models—after running the program, the student must be able to learn from the incorrect results; 4) explorations encourage students to pay attention to details; and 5) explorations have measurable results using pre- and post-tests. In this study's laboratory design, a similar set of guidelines were followed, with embedded instructional research in the laboratories. Each laboratory has explorative components that are graded as part of the activity worksheets, and student understanding is assessed as part of the laboratory pre- and post-tests.

Roumani outlined a set of design guidelines for an objects-first CS1 course [26]. The design principles that are similar include the following: 1) laboratories are not to be thought of as evaluation tools but seen as educational instruments that complement the coverage in lecture and in the textbook, where students should be allowed to discuss the tasks among themselves and/or seek help from the instructors; 2) the laboratories must be portable and self-paced; 3) laboratories must be explorative in nature where students have time to explore the correctness of a solution by writing tiny test programs; and 4) laboratories on object-based programming must be set in an abstraction that is credible and consistent. Each laboratory consists of three sections: explorative tasks, exercises, and checking. Each explorative task asks the student to look for some feature in a given specification, write a code fragment that uses (or implements) the feature, add debugging input/output (I/O), and then predict the output and verify it by actually running the code fragment. The checking section requires the student to write a main method that accomplishes a stated task and generate output with a specified format.

Chavey built a set of structured laboratories for CS1 [27]. Chavey's approach was similar to the design of this study, in terms of the overall design of the laboratories to teach aspects of CS and computer programming under the supervision of a laboratory instructor. However, the design goals differ. First, this study's laboratories are problem based or task based. Students are required to tackle a problem or accomplish a set of tasks and, through this process, are expected to learn the programming concepts and details. Second, Chavey's design did not include pre- and post-tests; the review of the laboratory design was based on the qualitative surveys of the students. Some conclusions included 1) the laboratories require significantly more time of the students, and 2) the laboratories significantly reduced the level of frustration in programming experienced by the students.

IEEE Proof

Some researchers have proposed using laboratory exams to test students [28]–[30] on their programming ability. For example, Califf and Goodwin proposed that a laboratory final exam was necessary since written exams in CS1 do not give instructors the needed perspective on the students' hands-on programming ability [29]. They found that the exam has been beneficial in improving students' attitudes toward programming and their proficiency in programming, and it has improved the instructors' ability to determine which students are sufficiently good programmers to continue on to CS2. Instead of a final exam, Barros *et al.* proposed a set of laboratory exams [30]. They found that laboratory exams clearly improved student motivation, shown by both their stated popularity and also by students' belief that assignments based on laboratory exams helped them achieve a better grade. Essentially, this study's laboratory design is similar to the latter example in which multiple, topic-specific laboratory exams are used. Designing small laboratory exams is also more modular and easier to maintain. Further, these exams pinpoint particular topics and allow for easier instructional research studies and analyzes.

To summarize, the approach to incorporating laboratories in introductory CS courses, as reported in this paper, is based on embedding instructional research design and assessment components into each laboratory to guide and motivate the design and development process in the way the pre- and post-tests and the activities were developed. For example, the pre- and post-tests included questions utilizing different levels of Bloom's taxonomy. The laboratory activities included explorations, reinforcement, and problem-based exercises. Each activity was graded based on a variety of questions—some required students to test their programs, to analyze their programs, to evaluate their programs, etc. Conceptual questions stemming from the laboratory activities and questions that required students to submit their programs for grading were necessary. The research team also designed the laboratories to incorporate cooperative learning.

B. Cooperative Learning

While direct instruction has been shown to be effective in certain domains, studies have shown cooperative learning to be an effective pedagogy for computer science, producing significant gains in student achievement [31]–[33]. Other advantages of cooperative learning are the development of communication and problem solving skills [34]. Because most students intend to join private industry, the goal of higher education is to prepare them. In private industry, collaboration and teamwork are the norm; therefore, collaborative learning in college settings better prepares students for what they will most likely encounter [35]. Direct instruction at the college level tends to emphasize individual skills and is often removed from environments encountered in industry [36]. Cooperative learning can also help students "become aware of the significance of small group dynamics as a tool for task achievement and success in a team environment" [31].

Cooperative learning is defined as, "working together to accomplish shared goals" [37]. The students are concerned not only with their own understanding of the material but also with that of the other group members. The students are working together for the same goal in cooperative learning; whereas, direct instruction lends itself to individual, *competitive* learning. This study relied primarily on the work of Johnson and Johnson [37] to model the implementation of cooperative learning in the CS1 laboratories.

For cooperative learning to be superior to individualistic competitive approaches, five elements are necessary: positive interdependence, face-to-face promotive interaction, individual accountability, interpersonal skills, and group processing [37]. Positive interdependence requires that group members "encourage and assist each other to do well" [38]. The students should feel that they would succeed or fail together. Face-to-face promotive interaction can be defined as individuals' encouraging and facilitating each others' efforts to achieve, complete tasks, and produce to reach the group's goals [38]. Individual accountability involves each group member providing his or her "fair share" of work and feedback. An interpersonal and small group skill is the group members' ability to interact and support one another positively. All five essential "elements" are included in the laboratory design.

A significant difference between cooperative learning and direct instruction is the role of the student. Direct instruction tends to involve an instructor disseminating information to the students while they passively take notes. On the other hand, cooperative learning allows the student to be an active agent in the learning process. Students are responsible in constructing knowledge while working with team members [35], [39]. This construction of knowledge is consistent with cognitive–developmental theory that poses that students learn better via collaboration and group discussions than by learning in isolation [40].

Another difference between cooperative learning and direct instruction is the role of the instructor. In a direct instructional setting, the instructor is responsible for disseminating information, while in a cooperative learning environment, the instructor serves more as a facilitator [41], [42]. In this role, the instructor helps with questions but only after the group has exhausted its attempt to answer questions.

III. Designs and Implementation of Laboratories

The process of redesigning the CS curriculum was preceded by extensive interactions between researchers from four academic departments: Computer Science and Engineering, Educational Psychology, Curriculum and Instruction, and Instructional Design. Much of the design was formalized through a joint seminar course organized in Spring 2003. The goals and objectives of this effort were spelled out, and a detailed plan was developed. The emphasis was not only on the development of novel approaches to deliver and assess course materials that promote "deep" learning, but also on developing a framework in which a systematic evaluation of the approaches and their short- and long-term effectiveness could be conducted. Therefore, cognitive and experimental psychologists and instructional designers were an integral part of this effort from the beginning. Because of the vast scope of this project, the research team decided to focus first on the CS1 course. In addition to lectures, students in the CS1 course attended a programming laboratory

IEEE Proof

that met for two hours each week. Approximately 25–30 students attended each laboratory section. The laboratories provided students with structured, hands-on activities intended to reinforce and supplement the material covered in the course lectures. Although brief instruction was often provided, the majority of the laboratory period was allocated to student activities.

The laboratories were designed by first selecting a set of core topics that could be covered during a semester (16 weeks). The laboratory topics were chosen based on lecture topics, modern software engineering practices, and Computing Curricula 2001 recommendations [2]. The focus of the laboratories was to promote problem-solving skills in students, while simultaneously providing them with hands-on experience in programming. The laboratory activities, for example, were designed to encourage individual investigation of problems and exploration of solutions.

A. Laboratory Design

The first step in developing the laboratories was to create a base document for each laboratory that included the following:

1) the laboratory's objectives;
2) prerequisite knowledge;
3) tools required;
4) instruction topics;
5) activities and exercises;
6) supplemental resources;
7) follow-on assignments;
8) relevance to course goals;
9) addressing Curriculum 2001 core topics;
10) ideas for pre- and post-test questions.

After a review of the base documents individually and collectively by the computer science faculty, each laboratory was developed by creating a series of five documents in parallel, including the following:

1) a student handout;
2) a laboratory worksheet;
3) an instructional script;
4) a pretest;
5) a post-test.

The student handout served several purposes. It was both the preparation guide and the laboratory script. Each handout included the laboratory objectives, a description of the activities that would be performed during the laboratory (including the source code where appropriate), a list of references to supplemental materials that should be studied prior to the laboratory, and a list of supplemental references that could be reviewed after the student completed the laboratory. The student handout also provided optional activities that could be completed during or following the laboratory to give students an opportunity for extra practice.

During each laboratory, students were expected to answer a series of questions for each activity and record their answers on a worksheet (paper). Worksheets contain questions specifically related to the laboratory activities and are intended to provide the students with an opportunity to find the answers through programming-based exploration. These worksheets also serve as an assessment tool to gauge the students' comprehension of topics learned and practiced in the laboratory.

In addition to the student handout, the laboratory instructor received an instructional script that provided supplemental material that may not have been covered during lecture, special instructions for the laboratory activities, hints, resource links, and useful insights. Additional space was provided at the end of the instructions for each activity to allow the instructor to record his or her comments regarding the activity and suggestions for improving the laboratory.

The laboratory pretests were online, and students were required to pass them prior to coming to laboratory; however, students were allowed to take each pretest as many times as necessary to achieve a passing score (80%). The pretest is open book and open note, and it included multiple-choice, short answer, and true/false questions. The goals of the laboratory pretest were to encourage students to prepare for the laboratory and to allow them to test their understanding of the laboratory objectives and concepts prior to attending the laboratory. Questions for the pretest were taken from a variety of sources, including the course textbook, other textbooks, and questions found on the Web. Questions were categorized according to Bloom's taxonomy [8].

During the last ten minutes of each laboratory, students completed an online post-test as another measure of their comprehension of laboratory topics. The test was open book and open note; however no collaboration was permitted. Like the pretest, questions were taken from a variety of sources and also categorized according to Bloom's taxonomy [8]. Notably, the goal of the pretest was to ensure that the students had read about and grasped the basic concepts for the laboratory, while the post-test was designed to assess how well they learned the concepts after they had performed the activities specifically designed to reinforce the concepts.

Table I shows the list of CS1 laboratories and their corresponding objectives. The first laboratory was designed to introduce the students to the overall computing environment in the CSE Department and the Integrated Development Environment (IDE) that they would be using. There were two event-driven programming laboratories; the first laboratory introduced the differences between event-driven programming and traditional sequential programming to students; the second laboratory addressed the capabilities and features of event-driven programming. There were three testing and debugging laboratories. The first laboratory introduced the idea of debugging to students and demonstrated how to debug using simple print statements and built-in features of the IDE and how to identify the different bugs. The second laboratory described a more systematic, holistic approach to debugging, with different strategies, such as a debug flag. The third laboratory introduced testing components from the viewpoint of software engineering, such as test cases and drivers. Overall, the three laboratories progressed from simple reactive debugging, to more goal-directed debugging, to standardized testing. The design emphasized testing and debugging, not only because the topic is of great importance to students, but also because it is a useful tool to help students learn other programming concepts and topics. For example, students

IEEE Proof

TABLE I
LIST OF LABORATORIES AND THEIR OBJECTIVES

Laboratories	Objectives: Students should be able to Ö...
Introduction to Integrated Development Environment (IDE)	• Log into the network using a UNIX machine and a CSE account. • Create a directory on UNIX, and traverse a UNIX directory structure. • Create and save a file on UNIX. • List the lab rules and hours of operation. • Understand the academic integrity policy in the Department of Computer Science and Engineering. • Know where to find help regarding CSE accounts and lab facilities. • Use an Integrated Development Environment (IDE) to create and update source code and compile and execute a sample program. • Use the on-line, hand in procedure to submit programming assignments and laboratory work.
Simple Class	• Define and explain basic object terminology including class, object, etc. • Identify the basic components of a Java program including private data member, public data member, public methods, private methods, constructor, etc. • Compile and execute a simple class. • Design and write a simple class.
Documentation	• Write clear, concise documentation describing a class and its members and methods. • Write Javadoc comments. • Use Javadoc to generate program documentation. • Determine when to use the three types of Java comments: single-line, block and Javadoc.
Testing and Debugging I	• Describe the difference between testing and debugging. • Describe the differences between syntax, semantic (logic) and runtime errors. • Debug a simple program using print statements. • Debug a simple program using the debugger functionality in BlueJ.
File Input/Output	• Write Java programs that perform primitive file operations including open, close, read, write, and check properties. • Write Java programs that verify a file exists, a file is readable and/or writable, and a file is a directory. • Write Java programs that perform file I/O using the FileOutputStream, FileInputStream, FileWriter, and FileReader classes. • Explain the principles of stream I/O and distinguish between byte-oriented and character-oriented streams.
Applets and Applications	• Distinguish between an application and an Applet. • Create an Applet with multiple methods. • Execute Applets using three methods (i.e., Applet viewer, browser, and application). • Explain when to use Applets to solve a problem. • List the advantages and disadvantages of Applets.
Event Driven Programming I	• Describe the difference between an event-driven model and a traditional sequential programming model. • Understand why event-driven programming is necessary and why it is used. • Understand the underlying principles and the meaning of event listening.
Exceptions	• Define what an exception is and why exceptions occur. • Explain the advantages of using Java exception handling over traditional error management techniques. • Distinguish between checked and unchecked exceptions. • Write a simple exception-handling routine for a single exception using *try*-*catch* and *try*ñ*finally* blocks. • Write methods that use the *throws* statement for exception handling.
Graphical User Interface (GUI) and Swing	• Apply basic design principles to GUI design. • Use the Java Swing interface to create simple GUIs containing panes, buttons, labels, combo boxes, and radio buttons. • Write GUI applications that handle events.
Event Driven Programming II	• Write a program with a single listener that handles multiple event *types* from a single event source. • Write a program with multiple listeners for a single event source. • Write a program with one listener for multiple event sources.
Testing and Debugging 2	• Use a variety of debugging strategies to identify bugs in programs. • Distinguish between a syntax error and a semantic error. • Use a debug flag to enable/disable debugging information.
Inheritance	• Explain what inheritance is, why it is used, and when it is used. • Identify the superclass and subclasses in a given program. • Understand the meaning of the keywords: *abstract*, *protected*, *extends*, and *super*, and know when and how to use them. • Write abstract classes and abstract methods. • Explain the override property of inheritance. • Define a reusable class based on inheritance. • Derive new subclasses from a superclass to extend a solution to new problems. • Design and give an inheritance solution for a problem.
Testing and Debugging 3	• Write a driver module to test a Java program. • Use Java assertions to check program logic. • Write and execute a set of test cases to test an application. • Determine what types of tests are necessary for each type of problem (e.g., invalid input data, boundary tests, branch testing, loop testing, etc.)
Recursion	• Identify a recursive method. • Identify the three basic elements of a recursive method. • Determine when a problem should be solved using recursion. • Given a recursive mathematical definition for a problem, write a recursive Java method to solve the problem.

IEEE Proof

learned about sorting and searching by finding and fixing the bugs within a program.

B. Instructional Research Design: Cooperative Learning

As mentioned in the previous sections, the approach to implementing CS1 laboratories included embedded instructional research design to study systematically the effect of the design on student learning. The design model is based on the work of Johnson and Johnson [37]. All five essential elements (positive interdependence, face-to-face promotive interaction, individual accountability, interpersonal skills, and group processing) were included in the laboratory design. The investigation focused studying two types of cooperative learning in the laboratories (structured cooperative learning utilizing the Johnson and Johnson model and unstructured cooperative learning similar to group work).

Note that a significant number of students enrolling in the CS1 course had not declared a major of study. Therefore, in the following studies, there is no breakdown analysis for CS majors and nonmajors.

1) Study: Effective Pedagogy for CS1 Laboratories: The purpose of the study was to determine the most effective pedagogy for CS1 laboratory achievement. According to the social constructivist view, the cooperative groups should perform higher than the direct instruction group. Previous research has shown significant differences between direct instruction and cooperative learning. The goal of this study was not only to compare direct instruction to cooperative learning but to compare it with different types of cooperative learning (structured versus unstructured).

Participants: The participants were 184 traditional undergraduate students from the University of Nebraska—Lincoln, many of whom have not declared their major of study. The study was conducted during Fall 2003, Spring 2004, and Fall 2004.

Procedures: The three laboratory structures used were: *cooperative group with structure* ($n = 55$), *cooperative group without structure* ($n = 65$), and *direct instruction* ($n = 64$), where n is the number of students in the group. The difference between the two cooperative groups was whether the structure of the group was formal or informal. Both cooperative groups consisted of three or four members. The cooperative structure group (formal) had defined roles, which alternated each week. The laboratory instructor was responsible for monitoring which student "drives" and which students review. The goal of this format was to develop interdependence among the group members based on the environment (shared computer) and breaking the tasks into smaller parts with each member responsible for a part. The group only functioned if each individual contributed his or her part for the whole group to complete their goal.

The cooperative unstructured group was similar to the cooperative structured group in that interdependence was created among the group members. The difference was that for this group format, the roles of the group members was not controlled. The members were responsible for assigning roles and completing tasks. In both cooperative groups, the laboratory instructor served as a facilitator giving both groups the freedom to solve problems themselves.

The last group format used in the study was direct instruction. This is the classical format in which students worked individually and competitively against other class members. This group served as the control group. The role of the instructor was to answer individual questions and discourage cooperation while students completed laboratory exercises.

The pedagogy assignment of each laboratory section (cooperative structured, cooperative unstructured, or direct instruction) was random. For the students enrolled in a section employing the cooperative approach, stratified random assignment was used to assign students to their cooperative groups. To accomplish this ranking, the placement test scores were used for this course. The scores were grouped into three categories: high, middle, and low. From each group, students were selected at random and placed in the cooperative group that they would be part of for the entire semester. This placement was to ensure heterogeneous grouping, which has been shown to be the most effective [37], [43].

The instructor's script for the laboratories was identical for all three groups to control for differences between groups. An important factor was that the laboratory instructor was the same for each group. The goal was to control as many variables as possible to allow the research team to make an informed decision as to which laboratory format was the most effective.

The laboratory groups that were manipulated were the cooperative groups. Both groups involved three to four classmates working together on their in-class laboratory exercises. The groups shared one computer with one student selected as the "driver," responsible for keying in the information on the computer, and the others as reviewers. The hypothesis was that the shared knowledge would promote higher-level thinking with improved problem solution.

Dependent Measures: The research team used laboratory final grades and individual laboratory post-test grades as the outcome measures. The combined outcome measures provided evidence of the effectiveness of laboratory pedagogy and achievement.

Total laboratory grades were measured by combining post-test and worksheet scores from each laboratory. Although some students worked in cooperative groups, all students were required to complete individually the post-test related to the topic covered in the laboratory. Details of the post-tests were described earlier in Section III-A. These tests were completed at the end of each laboratory.

Results: The first research question examined was student achievement in the laboratory. Analysis of variance (ANOVA) was used to determine significant differences between the sample means of cooperative groups with structure, cooperative groups without structure, and direct instruction. An ANOVA takes the variance (differences) among the three sample means and normalizes them using the variance (differences) within the groups accounting for sampling error. A significant result, probability (p-value) less than .05, indicates that the differences between the group means were something other than chance. Results from the ANOVA analysis were significant ($\mathrm{F}(2, 181) = 4.681, \mathrm{p} < .05$), with follow-up tests showing a significant achievement difference (as measured by final laboratory grade) between the direct instruction and cooper-

IEEE Proof

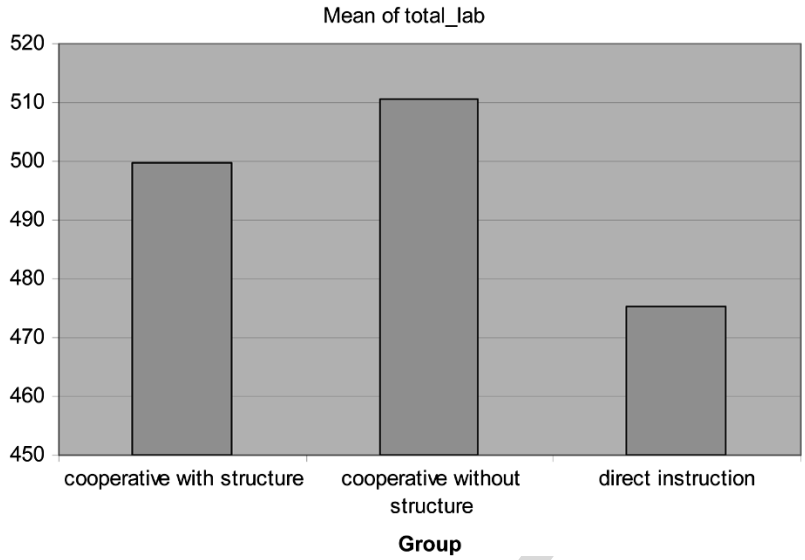

Fig. 1. Total laboratory scores (means) for the three types of groups: cooperative group with structure, cooperative group without structure, and direct instruction.

TABLE II
ANOVA STATISTICS COMPARING LABORATORY GRADES FOR EACH LABORATORY GROUP (COOPERATIVE WITH STRUCTURE, COOPERATIVE WITHOUT STRUCTURE, AND DIRECT INSTRUCTION) WITH THE GAMES–HOWELL POST HOC TABLE EXPLAINING THE SIGNIFICANT DIFFERENCES BETWEEN THE GROUPS

ANOVA -Total Lab

	Sum of Squares	Df	Mean Square	F	p
Between Groups	39905.04	2	19952.52	4.68	.01
Within Groups	771470.50	181	4262.27		
Total	811375.50	183			

Games-Howell

(I) group	(J) group	Mean Difference (I-J)	Std. Error	P	95% Confidence Interval ñ Lower Bound	95% Confidence Interval ñ Upper Bound
1	**2**	-10.58	10.97	.60	-36.70	15.55
	3	23.87	13.32	.18	-7.75	55.49
2	**1**	10.58	10.97	.60	-15.55	36.70
	3	34.45	11.29	.01	7.60	61.29
3	**1**	-23.87	13.32	.18	-55.49	7.75
	2	-34.45	11.29	.01	-61.29	-7.60

*. The mean difference is significant at the .05 level. Group 1 = cooperative structured; group 2 = cooperative unstructured; group 3 = direct instruction.

ative unstructured groups (see Table II). The Games–Howell follow-up test provides more specific detail as to which groups differed in the analysis (Fig. 1). Similar to the ANOVA, a significant result is determined by a p-value less than .05. The mean score difference between the cooperative structured group and cooperative unstructured group was not significant.

Discussion: Research has shown the importance of laboratories in computer science and the present study built on this premise. By manipulating the pedagogy used, cooperative learning proves to be the most effective learning approach. Surprising was that no significant difference existed between the structured and unstructured cooperative groups. The instructor did not control how the unstructured groups chose to define roles (driver versus reviewer), which may have led to the similar results between the cooperative groups. The significant differences between the cooperative unstructured groups and direct-instruction support cognitive–developmental theory indicated that work in groups is better than in isolation. The results provide enough evidence and reason to continue cooperative grouping in the CS1 laboratories.

2) Cooperative Learning: General Discussion: The studies provided the research team important information regarding the appropriate pedagogy for laboratories and the students' beliefs toward their CS skills. Qualitative responses from students in focus group interviews (five volunteer participants representing each laboratory format) provided insight into the student experience in CS1 laboratories. Themes found with the students representing each of the three laboratory pedagogies showed that the majority of the students preferred working in groups. The participants reported a "sense of community" and group problem solving: "Grouping us made us get to know other students which was nice." They also noted that the time required to complete assignments was decreased because the group worked together: "We had two hours to complete the laboratory activ-

IEEE Proof

ities and some times the two hours was not enough. It helped working in groups; otherwise we would never get it done." To be fair, not all students enjoyed working in groups. The major complaints were a sense of frustration working with others and the perceived effort of the group members. Some students felt that they were "smarter" students so that they were responsible for "carrying the load." Also reported was the perceived lack of self determination. One student in a cooperative group laboratory stated that "he preferred to work alone so that he could reap the rewards of his hard work." He wanted to be accountable only to himself. Participants in the direct instruction group complained of the "unfairness" of not being allowed to work with others. The laboratory instructor corroborated this conclusion and added that a sense of community was developed via grouping.

The interview also focused on self-efficacy and motivation. These comments were not concerned with pedagogy but the beliefs and experiences of each student. Emerging themes were that the students had very little idea of what would be expected of them. Computer science also proved to be much more difficult than they thought. Students stated that they had success in high school computer classes but that those classes were very different from what was encountered in CS1. This finding is consistent with anecdotal evidence from past semesters. Future goals of the reinventing CS Curriculum project are to integrate ACM2001 curriculum guidelines with high schools, to include further qualitative inquiry, and to improve the validity of the pretest measures. Sharing findings with high schools is important because the skills and expectations students bring with them to the university are greatly varied, and success in high school does not necessarily translate into success at the college level.

Implications: Note that although laboratories have been used in computer science courses in the past, their effectiveness has not generally been measured, nor has the way in which cooperative learning was used been reported. The goal of this study was not only to build on previous studies showing benefits of cooperative learning but also to determine the most effective approach of cooperative learning. The findings of this study have implications for CS laboratory pedagogy. The most significant is the result that shows cooperative groups attain higher achievement than the individual, direct instruction approach. These results show that the ACM2001 curriculum paired with the cooperative learning pedagogy not only produces higher achievement but also maintains consistency with the environment most students will find in private industry.

Finding quantitative differences between groups was a major goal of the research, but not the final one. In addition to measuring quantitative differences among the group formats, qualitative experiences were reported by interviewing selected students. The method used to obtain the qualitative data was focus groups in which themes were identified. This qualitative component provided additional depth and understanding to the research, allowing the research team to make the best decision on which laboratory format to use in future courses.

IV. Conclusion

This report has described an integrated approach using embedded instructional research design to design and implement closed laboratories for CS1, the first course in a typical computer science curriculum. Based on the vision of a more flexible and adaptive CS curriculum for students of different backgrounds and rapid changes in technology, this approach incorporated design, implementation, assessment, and evaluation. The Computer Science Department now has in place a process that guides laboratory design and implementation, collects data for studies based on both quantitative measurements and qualitative surveys, and allows for refinement of the laboratories as a result of the analysis. This paper shows how the research team has embedded cooperative learning into the laboratory design with positive results. Also revealed is how qualitative surveys are used to support the validation of the design and provide clues to subsequent laboratory revisions. The results of this study can be generalized to other structured engineering laboratories.

Based on the above results, changes have been made to the laboratories for CS1. Similar changes are also being designed and implemented into the CS2 laboratories using the same approach. The research team will continue to conduct studies to improve the validity and confidence of the results for CS2.

The research team will continue to "reinvent" computer science at the University of Nebraska—Lincoln based on evidence from these studies and other adjacent work. Cooperative learning pedagogy will be continued in CS1 laboratories based on the findings. Of interest is that the assignment of roles has not been a significant variable in the cooperative learning groups. Did the unstructured cooperative groups engage in formal assignment of roles? Are defined roles important? A mixed method approach is planned to answer these questions and build on the results obtained during the Fall 2003 and Spring 2004 semesters. Additional qualitative inquiry may help to identify and understand additional variables important to students. How do students perceive the quality of the laboratories in relation to what they need to learn in the course? Will changes to the curriculum or student beliefs improve overall course satisfaction? These are questions that need further investigation to add to the "reinvention" of computer science curriculum.

IEEE Proof

Acknowledgment

The authors would like to thank A. Zygielbaum for his support and C. Chen, S. Kasinadhuni, J. Bernadt, A. Kosenander, T. Fink, K. Halsted, X. Liu, J. Mallik, S. Das, S. Person, and Dr. C. Riedesel for their invaluable help in implementing this project.

References

[1] R. McCauley, W. Parrs, G. Pothering, and C. Starr, "A proposal to evaluate the effectiveness of closed laboratories in the computer science curriculum," *J. Comput. Sci. Colleges*, vol. 19, no. 3, pp. 191–198, 2003.

[2] XQXQXQ XQXQXQ, *ACM/IEEE Joint Task Force on Computing Curricula Staff, Computing Curricula 2001: Computer Science* ***<AUTHOR: Please provide author—ed.>***. Piscataway, NJ: IEEE Press, 2002.

[3] A. Samal, G. Nugent, L.-K. Soh, J. Lang, and S. Person, "Reinventing computer science curriculum at University of Nebraska," in *Technology-Based Education: Bringing Researchers and Practitioners Together**<AUTHOR: Please verify editor name for Zillig and provide city/state/country of publisher—ed.>***, P. Zillig, M. Bodvarsson, and R. Bruning, Eds: Information Age Publishing, 2005, ch. 10, pp. 203–224.

[4] P. J. Denning, D. E. Comer, D. Gries, M. C. Mulder, A. Tucker, A. J. Turner, and P. R. Young, "Computing as a discipline," in *Commun. ACM*, vol. 32, 1989, pp. 9–23.

[5] A. Tucker *et al.*, *Computing Curricula 1991: Report of the ACM/IEEE-CS Joint Curriculum Task Forc**<AUTHOR: Please provide publisher location—ed.>***: ACM Press, 1991.

[6] B. C. Parker and J. D. McGregor, "A goal-oriented approach to laboratory development and implementation," in *Proc. 26th SIGCSE Tech. Symp. Computer Science Education (SIGSCE'95)*, Nashville, TN, 1995, pp. 92–96.

[7] M. V. Doran and D. D. Langan, "A cognitive-based approach to introductory computer science courses: Lessons learned," in *Proc. 26th SIGCSE Technical Symp. Computer Science Education (SIGCSE'95)*, Nashville, TN, 1995, pp. 218–222.

[8] B. S. Bloom, B. B. Mesia, and D. R. Krathwohl, *Taxonomy of Educational Objectives (Two Vols: The Affective Domain & The Cognitive Domain)*. New York: David McKay, 1964.

[9] A. N. Kumar, "The effects of closed labs in computer science I: An assessment," *J. Comput. Sci. Colleges*, vol. 18, no. 5, pp. 40–48, 2003.

[10] M. Thweatt, "CS1 closed lab versus open lab experiment," in *Proc. 25th SIGSE Symp. Computer Science Education*, Phoenix, AZ, 1994, pp. 80–82.

[11] R. Lischner, "Explorations: Structured labs for first-time programmers," in *Proc. 32nd SIGCSE Tech. Symp. Computer Science Education (SIGCSE'2001)*, Charlotte, NC, 2001, pp. 154–158.

[12] J. Parker, R. Cupper, C. Kelemen, D. Molnar, and G. Scragg, "Laboratories in the computer science curriculum," *Comput. Sci. Educ.*, vol. 1, no. 3, pp. 205–221, 1990.

[13] S. R. Oliver and J. Dalbey, "A software development process laboratory for CS1 and CS2," in *Proc. 25th SIGCSE Tech. Symp. Computer Science Education (SIGCSE'94)*, Phoenix, AZ, 1994, pp. 169–173.

[14] J. K. Eaton. (1991) . Interactive software for self-paced instruction on laboratory instrumentation and computerized data acquisition. [Online]. Available: http://www.siggraph.org/education/nsfcscr/projects/intro/eaton.html

[15] B. Kurtz. (1993). An interdisciplinary, laboratory-oriented courses sequence for computer-based problem solving. [Online]. Available: http://www.siggraph.org/education/nsfcscr/projects/intro/kurtz.html

[16] J. C. Prey. (1995). Cooperative learning in an undergraduate computer science curriculum. [Online]. Available: http://fie.engrng.pitt.edu/fie95/3c2/3c23/3c23.htm

[17] K. Robbins *et al.*, "Solving the CS1/CS2 lab dilemma: Students as presenters in CS1/CS2 laboratories<," in *Proc. 32nd SIGCSE Tech. Symp. Computer Science Education*, 2001, pp. 164–168.

[18] L.-K. Soh, A. Samal, S. Person, G. Nugent, and J. Lang, "Analyzing relationships between closed labs and course activities in CS1," in *Proc. 10th Annu. SIGCSE Conf. Innovation Technology in Computer Science Education (ITiCSE 2005)*, Monte de Caparica, Portugal, Jun. 27–29, 2004, pp. 183–187.

[19] ——, "Closed laboratories with embedded instructional research design for CS1," in *Proc. 36th SIGCSE Tech. Symp. Computer Science Education (SIGCSE 2005)*, St. Louis, MO, Feb. 23–27, 2004, pp. 297–301.

[20] D. T. Joyce, "A virtual lab to accompany CS1 and CS2," in *Proc. 21st SIGCSE Tech. Symp. Computer Science Education*, Washington, DC, 1990, pp. 40–43.

[21] K. C. Ueberroth, "Managing high-volume computer literacy labs," in *Proc. 22nd Annu. ACM SIGUCS Conf. User Services*, Ypsilanti, MI, 1994, pp. 301–307.

[22] J. L. Hein, "A declarative laboratory approach for discrete structures, logic and computability," in *ACM SIGCSE Bull.*, vol. 25, 1993, pp. 19–25.

[23] R. Bruce, J. D. Brock, and K. Bogert, "X-lab: XML-based laboratory exercises for CS1," in *Proc. 42nd ACM Annu. Southeast Regional Conf.*, Huntsville, AL, 2004, pp. 434–435.

[24] R. Ross, "Education forum: Lecture and lab syllabus for a breadth-first introductory computer science course sequence following the data structures and algorithms paradigm," in *ACM SIGACT News*, vol. 25, 1993, pp. 38–43.

[25] R. Geitz, "Concepts in the classroom, programming in the lab," in *Proc. 25th SIGSE Symp. Computer Science Education*, Phoenix, AZ, 1994, pp. 164–168.

[26] H. Roumani, "Design guidelines for the lab component of the objects-first CS1," in *Proc. 33rd SIGCSE Tech. Symp. Computer Science Education (SIGCSE 2002)*, Covington, KY, 2002, pp. 222–226.

[27] D. Chavey, "A structured laboratory component for the introductory programming course," in *Proc. 22nd SIGCSE Tech. Symp. Computer Science Education (SIGCSE 1991)*, San Antonio, TX, 1991, pp. 87–295.

[28] A. T. Chammilard and J. K. Jointer, "Using lab practical to evaluate programming ability," in *ACM SIGCSE Bull.*, vol. 33, 2001, pp. 159–163.

[29] M. E. Califf and M. Goodwin, "Testing skills and knowledge: Introducing a laboratory exam in CS1," in *33rd SIGCSE Tech. Symp. Computer Science Education (SIGCSE 2002)*, Covington, KY, 2002, pp. 217–221.

[30] J. P. P. Barros *et al.*, "Using lab exams to ensure programming practice in an introductory programming course," in *Proc. 8th Annu. Conf. Innovation Technology in Computer Science Education*, 2003, pp. 16–20.

[31] A. Joseph and M. Payne, "Group dynamics and collaborative group performance," in *Proc. 34th SIGCSE Tech. Symp. Computer Science Education (SIGCSE 2003)*, Reno, NV, 2003, pp. 368–371.

[32] T. Gatfield, "Examining student satisfaction with group projects and peer assessment," *Assessment Evolution Higher Educ.*, vol. 24, no. 4, pp. 365–377, 1999.

[33] M. Malinger, "Collaborative learning across borders: Dealing with student resistance," *J. Excellence College Teaching*, vol. 9, no. 1, pp. 53–68, 1998.

[34] Z. Qin, D. Johnson, and R. Johnson, "Cooperative versus competitive efforts and problem solving," *Rev. Educ. Res.*, vol. 65, no. 2, pp. 129–143, 1995.

[35] K. A. Yerion and J. A. Rinehart, "Guidelines for collaborative learning in computer science," *SIGCSE Bull.*, vol. 27, no. 4, pp. 29–34, 1995.

[36] J. C. Prey, "Cooperative learning in an undergraduate computer science curriculum," presented at the ASEE/IEEE 1995 Frontiers in Education Conf.**<AUTHOR: Please provide location and full date of conference or pages if in proceeding—ed.>**, 1995.

[37] D. W. Johnson and R. T. Johnson, *Learning Together and Alone: Cooperative, Competitive, and Individualistic Learning*, 3rd ed. Upper Saddle River, NJ: Prentice-Hall, 1991.

[38] M. Jensen, D. W. Johnson, and R. T. Johnson, "Impact of positive interdependence during electronic quizzes on discourse and achievement," *J. Educ. Res.*, vol. 95, pp. 161–166, 2002.

[39] C. M. Keelerand and R. Anson, "An assessment of cooperative learning used for basic computer skills instruction in the college classroom," *J. Educ. Comput. Res.*, vol. 12, no. 4, pp. 379–393, 1995.

[40] L. S. Vygotsky, *Mind in Society: The Development of Higher Psychological Processes*. Cambridge, MA: Harvard Univ. Press, 1978.

[41] M. S. Meloth and P. D. Deering, "The Role of the teacher in promoting cognitive processing during collaborative learning," in *Cognitive Perspectives on Peer Learning*, A. M. O'Donnell and A. King, Eds. Mahwah, NJ: Lawrence Earlbaum, 1999, pp. 235–255.

[42] E. G. Cohen, "Restructuring the classroom: Conditions for productive small groups," *Rev. Educ. Res.*, vol. 64, pp. 1–35, 1994.

[43] N. M. Webb and A. S. Palinscar, "Group Processes in the Classroom," in *Handbook of Educational Psychology*. New York: MacMillan, 1996.

IEEE PROOF

Jeff Lang is currently working toward the Ph.D. degree in educational psychology at the University of Nebraska—Lincoln.

He is a Graduate Research Assistant at the National Center for Information Technology in Education at the University of Nebraska—Lincoln. He is also a visiting Professor at Nebraska Wesleyan University**<AUTHOR: city?—ed.>**. His research interests include cooperative learning, self-efficacy, and note-taking functions.

Gwen C. Nugent is a Research Associate Professor at the National Center for Information Technology in Education at the University of Nebraska—Lincoln. She coordinates development and research projects focusing on the impact of technology to improve student learning and teacher competencies, with special emphasis on multimedia instruction and online assessment. She has over 30 years experience in the design, production, and evaluation of mediated instruction and has served as project manager for over 300 multimedia projects. The majority of these projects are distributed nationally and internationally, and many have won national awards for their educational impact and effectiveness.

Ashok Samal received the Bachelor of Technology degree in computer science from the Indian Institute of Technology, Kanpur, India, in 1983 and the Ph.D. degree in computer science from the University of Utah**<AUTHOR: city?—ed.>**.

He is an Associate Professor with the Department of Computer Science and Engineering at the University of Nebraska—Lincoln. His current research interests include computer science education, image analysis, geospatial computing, and data mining.

Leen-Kiat Soh (S'90–M'97) is Harold and Esther Edgerton Assistant Professor with the Department of Computer Science and Engineering at the University of Nebraska—Lincoln. His research interests are in multiagent systems, intelligent education systems, machine learning, and computer science education. He has been involved in a departmentwide Reinventing Computer Science Project at the University of Nebraska—Lincoln and has built two intelligent educational applications. He has published more than 50 journal, conference, and workshop papers in the areas of remote sensing, image processing, multiagent systems, and machine learning.

Dr. Soh is a Member of the Association for Computing Machinery (ACM) and the AAAI.

IEEE Proof