Abstract

YANG, KAI. Pair Learning In Undergraduate Computer Science Education. (Under the direction of Laurie A. Williams)

Anecdotal evidence in industry and academia has shown that pair programmers, whereby two programmers work collaboratively on the same task on one computer, can produce higher quality design and code than solo programmers who work alone. Educators in various fields have also found that collaborative work helps students learn better and allows them to be more confident in their study. Based on these two findings, pair learning, which is the practice of pair programming with students, was generated and applied in a computer science class setting.

To validate the effectiveness of pair learning, an experiment was run in a CS1 course at North Carolina State University in the fall semester of 2001. The experiment focused on freshman and sophomores. Approximately 120 freshman and sophomores participated in the study. The students registered in two sections of a CS1 course without knowledge of the experiment. Both sections had the same instructor, same programming assignments and same examinations. One section utilized a traditional education style whereby all students worked alone; the other section followed the pair learning paradigm where all students working with a partner. Results supported the following findings:

1. Students who followed the pair learning style performed better on the programming assignments and were more likely to get a higher score on the examinations.

2. A higher percentage of the students who utilized the pair learning style succeeded in the CS1 class by completing the course with a grade of C or better.
3. The teaching assistants in the pair learning section had reduced workload, since students in the pair learning section were more self-sufficient.
4. Paired students demonstrated higher-order thinking skills than the students when compared with the students who worked alone.
5. Students in the pair learning section were more likely to think far beyond the programming assignment to applying their knowledge in more challenging programming contexts.
6. Paired students collaborated more extensively with each other; and collaboration is an important skill for programmers in industry.

Pair Learning In Undergraduate Computer Science Education

by

Kai Yang

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

COMPUTER SCIENCE

Raleigh

2002

APPROVED BY:

Annie I. Antón, Ph.D.   Eric N. Wiebe, Ph.D.

Laurie A. Williams, Ph.D.
Chair of Advisory Committee

BIOGRAPHY

Kai Yang received a Bachelor of Science degree in Computer Science department from Nankai University, China in 1998. After graduating, she stayed in the same university as an instructor due to her outstanding performance. Having taught one year, she came to US to join her husband in 1999. In Fall 2000, she enrolled in the Computer Science department in North Carolina State University.

Acknowledgement

This thesis would not have been possible without the help and support of the following faculty, colleagues, friends and family. I begin by thanking my advisor, Dr. Laurie Williams who provided immeasurable assistance throughout my research. As English is not my first language, my thesis required a good bit of polishing and she edited my thesis so carefully, even the grammatical errors. I appreciate her assistance and hope it comforts her to know that I learned a lot from this process. Laurie is not only my advisor; she is like an older sister. When I met a problem in life, she kindly to helped me and I am most grateful. There is a Chinese proverb that characterizes my two years at NC State: “stay with good people, you can learn good things.”

My committee members, Dr. Eric Wiebe and Dr. Annie I. Antón were very generous and patient with me. Dr. Wiebe provided a great deal of statistical analysis assistance that proved extremely valuable; Dr. Antón polished my thesis and edited it.

Ms. Carol Miller and Ms. Suzanne Balik, the instructors of the CS1 course within which I was conducting my research, were kind and supportive of my research.

Miriam Ferzli, a Ph.D. candidate in Mathematics, Science, and Technology Education department, participated in this research by providing detailed student observations that were very enlightening.

The teaching assistants worked hard in the labs and the students provided me with valuable feedback; they were central to this study.

I thank my colleagues with whom I enjoyed working: Ashish, Mayank, Vinay, Boby, Harman, David, Puneet, Anuja and Prashant.

Special thanks go to my family in China, my Mom, Dad and younger brother. They asked me about my study progress each time we spoke, always encouraging me to persevere. Their trust and encouragement were my source of strength and inspiration as I completed my research and wrote this thesis. Finally, I thank my husband for constantly supporting me during the challenging times and for sharing in my happiness during the times marked by significant progress. Without him, this thesis would not have been possible.

Table of Contents

Page

LIST OF TABLES

Page

LIST OF FIGURES

Chapter 1 Introduction

1.1 Research Motivation

"Technology alone doesn't make for good teaching and certainly does not guarantee good learning. …… one must consider how basic, pedagogical principles and puzzles associated with sound teaching practice might (or might not) apply to each medium."

------ Al Andrade, 2002

1.1.1 The Need for More Computer Science Students

With the increasingly popularity of computer technology, more and more technologists in computer science are required in industry. Since universities are the major source of propagating computer skills and knowledge, it is critical to help students be as successful as possible in computer science courses. At many universities, students need not declare their major during their first year. As a result, many college students take computer science classes before they decide their major. Therefore, the introductory computer courses play an important role in students' ultimate decision to pursue a computer science degree.

1.1.2 Success Rate in the Beginning Computer Science Course

Educators generally measure the success rate of their classes in terms of the percentage of students who complete the course (i.e. do not withdraw) with a grade of C or better. Often this statistic is expressed in terms of the converse, the "DFW" rate. The DFW rate is the percentage of students in a class who receive a grade of D or F or withdraw from the class. Anecdotally, many educators communicate a typical DFW rate of about 50% in CS1 classes. The problem is often caused and/or exacerbated by the fact that many students have a hard time seeking help when they do programming assignments. Alternately, some students want to save face by not asking questions they think may be perceived as “stupid”. Thus, computer science educators are challenged to increase the overall success of introductory classes, to evoke students’ interest, to help them learn better, and in turn to decrease the DFW rates.

1.1.3 Pair programming

Pair programming is a style of programming where two programmers work side-by-side at one computer, continuously collaborating on the same design, algorithm, code or test. One individual is called the “driver,” he/she is in charge of the keyboard and mouse and is recording a design or code implementation. The other is called the “navigator”, and continuously observes the driver while the driver programs or does the design. It is the navigator’s responsibility to pick out the tactical and strategic defects. Tactical defects are syntax errors, typos, calling the wrong method, etc. Strategic defects occur when the driver is headed down the wrong path – what they are implementing just won’t accomplish what it needs to accomplish (Williams, Wiebe et al. 2002). The navigator also thinks strategically about the work’s direction. During their collaboration, the driver and navigator can discuss the direction of the program, the best algorithm to use, or anything that may help solve the problem at hand. The brainstorming helps the pair finishes in a faster and more effective way. We believe the ample communication makes the pair programming relationship very effective.

Pair programming has been used sporadically for decades (Williams and Kessler 2002). More recently, pair programming has been popularized as an important practice of an emerging software development methodology, eXtreme Programming (XP) (Beck 2000). XP was developed initially by Smalltalk code developer and consultant Kent Beck with colleagues Ward Cunningham and Ron Jeffries. “It is a lightweight discipline of software development based on principles of simplicity, communication, feedback, and courage” (Jefferies 1999). The XP methodology is growing in popularity, though its great success has been reported anecdotal rather than empirically.

1.1.4 Pair Learning

Computer science courses require a great deal of programming assignments. Pair learning introduces the pair programming technique in computer science education so that students learn to program by working with another student at one computer.

1.1.5 Motivation of this research

As we stated in Chapter 1.1.2, one of the obstacles that prevent students from getting a grade of C or above is the fact that many students have hard time seeking help when they do programming assignments, or they are reluctant to ask others. In response, researchers considered using pair learning to help students help each other in a non-threatening way. With pair learning, two students collaborate on one task on one computer; they can discuss between each other. The problem of waiting for teaching assistant for help can be alleviated. Since the students are peers, they will not have as much fear of “losing face” or appearing “stupid” to each other.

The objective of this research is to assess and refine the pair learning technique in computer science education. Specifically, we examine whether the practice can affect the performance and attitudes of students as they advance through the computer science curriculum. We assess the following hypotheses:

1. More students that have participated in pair learning in CS1 will succeed in the class by completing the class with a grade of C or better.

2. Students' participation in pair learning in CS1 will lead to better performance in examinations (exams are completed solo by all students) in that class.

3. Students' participation in pair learning in CS1 will lead to better performance on course projects in that class.

4. Students' participation in pair learning in CS1 will lead to a more positive attitude toward the course and toward computer science in general.

5. Students' participation in pair learning leads to a lower workload for the teaching staff.

1.2 The Research Approach

To assess the effectiveness of pair learning, a formal experiment was performed at North Carolina State University in Fall 2000 with a CS1 course. In the experiment, students were divided into two sections; one section learned in the traditional way, while the other section utilized pair learning. Students registered for their section without knowledge of the experiment. Both sections had the same instructor, same examinations and same project assignments. At the end of the semester, we compared the two groups to investigate our hypotheses.

1.3 Summary of Remaining Chapters

The remainder of this thesis is organized as follows:

Chapter 2 A SURVEY OF RELATED WORK reviews the prior relevant research conducted by other educators and researchers.

Chapter 3 METHODOLOGY describes the details of the experiment we ran in the CS1 "Introductory to Java" course at North Carolina State University.

Chapter 4 EXPERIMENT FINDINGS discusses and analyzes the quantitative and qualitative results we derived from this experiment.

Chapter 5 CONCLUSION AND DISCUSSION summarizes our conclusions and discusses possible avenues for future.

APPENDIX introduces the tools we implemented to aid our study, including a web-based pair-assign program and a web-based peer-evaluation program. The Appendix also contains copies of the questionnaire and programming assessment instrument utilized in our study.

Chapter 2 A Survey of Related Work

Many educators and researchers have made contributions to help students learn better. This chapter surveys the studies and contributions of educators in areas related to pair learning.

2.1 Active Learning

Active learning is an educational method that is different from traditional education. In the traditional approach to college teaching, most class time is spent with the professor lecturing and the students passively watching and listening. With active learning, students actively take part in the class besides listening and taking notes; the instructor's responsibility is to direct the students to learn themselves (Lorenzen 2002).

A simple implementation of the active learning technique is to insert some short breaks in the traditional lecture. According to Bonwell (Bonwell 1996), there are five variations of activities that can be done during these breaks: the pause procedure, short writes, think-pair-share, formative quizzes, and lecture summaries. The pause procedure is to stop for a while every 13 to 18 minutes in the lecture, allowing the students to discuss among them or catch up with the notes they missed. With short writes (Angelo and Cross 1993), the instructor asks some simple questions which ask the student to summarize the material taught in class, and the students write answers to these questions (answers are not graded). Think-pair-share occurs the when the instructor asks a question of the class. Students think individually about the answer first, and then share their answers among neighbors. Formative quizzes are some short quizzes with the questions similar to those seen on exams, but the quiz will not be graded. Lecture summaries invite the students to summarize the lecture so that they can have a global view of the lecture.

Ruhl et al. (Ruhl, Hughes et al. 1987) performed an experiment to examine the effectiveness of the pause procedure. In the control group, students learned in a traditional way. In the experiment group, students employed a "pause procedure" every 12-18 minutes. During the pauses, students were allowed to work in pairs for two minutes to discuss and make up their notes. The instructor did not participate in the students' discussions. Both the experimental and control groups were asked to write down everything they remembered in class for three minutes at the end of each lecture (free-recall). Twelve days after the lecture, the students were also given a 65-item multiple-choice comprehensive test to measure long-term retention. Their research lasted two semesters and the results showed that the experimental group students performed significantly better on the free-recall quizzes and the comprehensive test. In addition, the difference between the two groups in the mean scores was large enough to make an up to two-letter grades difference.

Dr. Richard M. Felder (Felder 1995);(Felder, Felder et al. 1998) has performed a longitudinal study in the Chemical Engineering department at North Carolina State University. He performed an experiment in the Fall 1990 semester in an introductory chemical engineering course in which 123 students were enrolled. With the experimental group, he used several active learning techniques. He generally did not teach the class for more than 15 minutes without giving an exercise of some sort or a stretch break. Occasionally, he ended a period with a "one-minute paper,"(Angelo and Cross 1993) from which he elicited responses and tailored his lecture accordingly. In “one-minute paper”, at the end of class, the instructor stops the students two or three minutes early and asks students to respond to some variation on the two questions briefly, “What was the most important thing you learned during this class?” and “What important question remains unanswered?” By performing this, on one hand, students can recall the class and review it to answer each question, meanwhile they can learn how experts distinguish main points from the details from the feedback from instructors to their answers; on the other hand, instructors can decide whether any mid-course corrects or changes are needed, which will help the instructors teach better next time. Additionally, Felder asked the students to complete the homework in teams. As comparison, he picked a control group, which was constituted of 189 students enrolled in the same class in the Fall 1992 semester. The control group was taught in the traditional way. He hypothesized that the experimental group will learn better, i.e. have higher retention rate, more positive attitudes and greater confidence levels in their problem-solving skills. To assess the hypothesis, he collected a great deal of data including demographic data, SAT scores, first-year grade-point averages, etc. as their background information; course grades, statistics on persistence in the chemical engineering curriculum, etc. as their subsequent information. His study indicated that the experimental group outperformed the comparison group on the retention rate, graduation in Chemical Engineering and many more graduates chose Chemical Engineering as their major to pursue advance degrees.

Active learning is a teaching strategy. It can refer to anything students do in a class other than listening to the instructor and taking notes. It includes asking and answering questions, discussing and explaining concepts, formulating and solving problems, summarizing lectures, brainstorming lists, and many other activities. The work may be done by individual students or students working in groups. (Haller, Gallagher et al.)

2.2 Cooperative Learning

Cooperative learning is another instructional methodology. In cooperative learning, students work in a team to accomplish a common goal (Slavin 1990).

MacGregor (MacGregor 1998) investigated an instructional approach that emphasized both collaborative techniques and structured design concepts. The students were taught programming in an individualistic-traditional or a team-structured way. He performed an experiment with thirty-two high school students (juniors and seniors) enrolled in two sections of a computer course. The two sections were taught by the same instructor. His results revealed the team-based structured programming methodology significantly outperformed the individualistic-traditional methodology when comparing students' programming performance and attitude toward programming. He also found that this approach had significant positive correlations between students' ability to design programs and their overall programming performance and attitude toward programming. He provided some qualitative results: the team-based groups spent more time on the planning phase and discussions happened among them; the individual group spent more time debugging their programs. The design phase is very important in the programming; spending more time in design phase will generally save considerable time in debugging. MacGregor’s study confirmed Lemos’ study (Lemos 1979) that team-based approaches save time in debugging programs.

Priebe (Priebe 1997) also performed research on second-semester university computer science students. One group of students received the traditional lecture while the other group participated in cooperative learning for the same number of hours as the previous group. His three hypotheses follow: (1) the cooperative learning students would do better in concept comprehension than the control group; (2) cooperative learning students would have improved logical thinking skills than the control group students and (3) the cooperative learning students would have better class attendance rates. He used statistical methods to test his hypotheses. The results revealed no difference between the cooperative learning and control groups in concept comprehension or logical reasoning ability. However, the cooperative learning group did have significantly better attendance. Priebe also emphasized how team-oriented activities in the classroom modeled real-world teamwork in industry. He believes that the group setting fostered positive peer pressure that led to neater and more complete assignment submissions. He, too, commented on the higher level of self-teaching that occurred in the cooperative group setting.

Cooperative learning is a special form of active learning. In cooperative learning, students are assigned to achieve one goal together. This could happen either in class or outside of class.

2.3 Pair Programming

Pair programming is a form of cooperative learning, since in pair programming, two students work in a team to achieve a common problem.

2.3.1 Prior Research in pair programming

In 1998, Nosek performed an experiment with 15 experienced system programmers to work on a challenging problem (Nosek 1998). Five programmers worked individually, and 10 programmers worked in pairs. He hypothesized the following: 1) programmers working in pairs will produce more readable and functional solutions; 2) pair groups will take less time on average to solve the problem; 3) pair programmers would be more confident about their work; 4) more experienced programmers would perform better than less experienced programmers. He employed several statistical tests to assess his hypotheses. The programmers working in pairs experienced a higher readability and functionality score. They were also more confident and enjoyed the programming more than the group comprised of individual programmers. In other words, hypotheses 1, 3 and 4 were supported. On average pair programmers spent less time than the individual programmers, though the result is not statistically significant.

Although Nosek's study showed that pair programmers performed better than individual programmers, his study was somewhat narrow in scope. He drew his conclusions from only 45 minutes of pair programming time with only 15 experienced programmers. Williams (Williams 2000), performed a larger pair programming experiment; she studied 41 senior software engineering students in a software engineering class. In that experiment, she divided the 28 students to form the collaborative/pair groups and 13 students to be in the group comprised of individual programmers. Student grade point averages (GPA) were used as a basis to ensure the groups were academically equivalent. More tasks were assigned to students in the paired group to ensure that the overall workload was even between the both experimental subject groups.

She measured the time each group spent to finish the given task as well as the code quality. In Figure 1, we see that as a pair, the time pair programmers spent is less than that by individual programmers. But when we counted the pair as two people, then we see in program 2 and program 3, the pairs used essentially the same amount of time to finish the task. In program 1, since the students collaborated with each other the first time, they had a "jelling" period, when they need to become familiar with each other and with working in a pair. As a result, the pairs took 60% more time than the individuals.

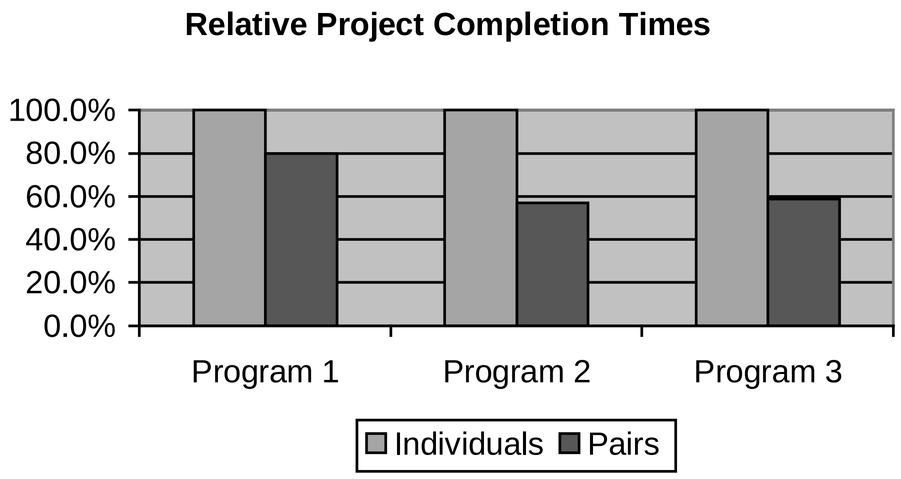


Figure 1 Williams et al: Relative Project Completion Times

Although the pairs did not outperform the individuals in the time, they did a better job in achieving higher quality code than the individual groups. They had a higher rate of passing the automated tests made by the teaching staff. The figures are listed in Table 1:

Table 1 Williams et al: Percentage of Test Cases Passed

	Individuals	Pairs
Program 1	73.4	86.4
Program 2	78.1	88.6
Program 3	70.4	87.1
Program 4	78.1	94.4

The experiment indicated that pair programming could produce higher quality code than the individual programmers given around the same amount of time.

More recently, educators at the University of California-Santa Cruz have performed a pair programming experiment in an introductory computer science undergraduate programming course (Bevan, Werner et al. 2002). They have also found that the experimental group students, who followed the pair programming technique had a higher retention rate and their performance on programming assignments was better than the control group (McDowell, Werner et al. 2002).

2.3.2 Pair programming factors

There are several factors that come into play in pair programming: pair pressure, pair share, pair review, pair jelling, pair-think, pair-relaying (Williams 2000). We focus our discussion on the first four factors below.

2.3.2.1 Pair Pressure

Pair programming places a positive “pair pressure” on each team member in the pair (Williams, Kessler et al. 2000). From the observations we had in our experiment, we saw all the student pairs working all the time: the driver typed the code; the navigator was alert and helping. None of them checked their email or did something unrelated to the lab, since no one wanted to let their partner down (Williams 2000).

Meanwhile, from the feedback the pairing programming students gave, a lot of the students were prepared before they come to the lab. As one pairing student said “I don’t want to let my partner down”. Both Nosek and Williams’ studies indicated similar findings.

2.3.2.2 Pair Share

“Put a less experienced person together with a more experienced person, and the former will be more likely to stretch.” (Leuf and Cunningham 1999)

In the experiment we conducted in the study, some students learned programming when they were in high school; others did not have any knowledge about programming. So when they paired up with each other, a lot of students provided feedback that they learned a lot from the other partner. They wanted to work or collaborate with their partner again and/or they enjoyed working with each other. One student wrote, “It was a great pleasure having [name] as a lab partner. She shared her past experiences with Java programming with me which helped me understand programming and compiling a lot better.” Another said “[name] was always prepared, and because he had previous experience, he helped me learn new things." An interesting thing is that students who were said to be more experienced did not realize they were only outputting. One of such a student said, "My partner also helped me point out something I was wrong." This may confirm the experiences Jeffries (Williams, Kessler et al. 2000) reported. He worked with a least-experienced developer, while at the end he felt the junior programmer helped him.

2.3.2.3 Pair Reviews

"As a testimonial, in the last six months before launching, the only code that caused problems was code written solo." --- (K. Beck, 1999)

Review methods can be described as inspections, walk-throughs, and personal reviews. Humphrey (Humphrey 1995) states "Doing reviews is the most important step you can take to improve your software engineering performance." When programmers work in pairs, one of the responsibilities of the navigator is to review the code the driver is writing. If he or she finds any errors and has different opinions, they will discuss it immediately. So in pair programming, the navigator reviews the code continuously. As a result, the code written by pair programmers can be considered to have been reviewed.

2.3.2.4 Pair Jelling

In pair programming, two programmers work together on one task. They are like one, since they need to "unify" their thoughts. They are also equal, since they switch "driver" and "navigator" roles often.

But in the normal case, the programmers have worked alone for a while. So as two individuals, when they work together, they need to spend some time to adjust to the other’s working habits, programming style, etc (Williams, Kessler et al. 2000). In industry, the adjustment period is historically hours or days, depending on the individuals. In the university, we could also see from Williams’ study above, for the first assignment, the pairs finished in shorter elapsed time and had better quality, but their total working hour is 60% more than the solo group. This is also true in our study, for the project assignments and the exams, the paired students outperformed more than the solo students in the second time over the first time.

Chapter 3 Methodology

3.1 Overview

In the Fall 2001, a formal experiment was run in a CS1 course, Introduction to Java, at North Carolina State University. The purpose of this study was to assess the efficacy of pair learning for improving students' learning success including course retention rate, examination performance, project performance and attitudes of computer science. The experiment involved three steps. First, the students registered for one of the two CS1 course sections. In one section, students utilized the pair learning technique and worked in pairs in the laboratory; in the other section, students worked alone in the laboratory. When they signed up for the course section, students had no knowledge of the experiment. In step two, we collected all the students' data, including their ten lab assignment scores, three exam scores, three project scores and other background information, such as their gender, SAT-Math score, and programming assessment score. The programming assessment was a short quiz with eight questions that was given during the first lab period. It will be introduced in detail in Section 3.3.2. In last step, statistical analysis was performed on the data to evaluate the pair learning effect on the students.   The first two of these steps will be discussed in this chapter. The data analysis will be discussed in Chapter 4.

3.2 Step One: Class Registration and Pair Assignment

3.2.1 Class Description

The course was taught with 50-minute lectures twice a week and one three-hour lab each week. Students attended labs in groups of 24 or less with other students in their own lecture section. The labs were closed labs, whereby students did their assignments in the allotted time and place. The teaching assistant often started out with a short lecture on the assignment and was then available for questions during the lab period. There were two midterm exams, one final exam, ten lab assignments, and three programming projects. The programming projects were primarily done outside of the closed lab.

The course is a service course, and is therefore taken by students throughout the university and life-long students from industry to improve their programming skills. Most students are from the College of Engineering. Additionally, most students are freshmen and sophomores.

In the Fall of 2001, there were 112 students enrolled in the solo lecture section (later called “section”), and 87 in the paired lecture section (later called “section”). For each course section, students registered into five lab sections. There were five paired and five solo laboratory sections in total. Both lecture sections took the same class with the same instructor at different times, had the same lab assignments, the same programming projects and the same midterms and finals. (The exams were given to the second section immediately after the first, leaving little time for students to tell the second section about the exam content.) If the total number of the students in a paired lab was odd, then a group of three students would work together: no students worked alone. If a student’s partner did not show up after ten minutes of the lab starts, the student would be assigned a new partner by the teaching assistant.

The students were assigned to collaborate with a new partner every two or three labs. Two benefits of this switching are that students would not get tired of or have ongoing difficulty interacting with a particular partner and they could collaborate with several students throughout the semester. Additionally, the teaching assistant could evaluate the contributions of each student based on the opinions of four partners (see Section 3.2.2). For later reference convenience, we call each period with the same partner a “rotating cycle”. The rotating cycle of the 10 labs were: lab 1 – 3, lab 4 – 6, lab 7 – 8 and lab 9 – 10. A web-based partner assigning tool was built as part of this research; the students did not choose their own partner. This tool will be explained in detail in Appendix B.

Before each lab, the teaching assistants went to the pair assign website to obtain the pair assignment for the students in their section(s). During the lab period, the teaching assistants made any changes according the students’ presence. For example, he or she can re-assign pairs and mark the students who are absent. At the end of the lab, the students are required to turn in their lab assignment, and the teaching assistant will give a score of 0 – 100 according to the program’s performance.

3.2.2 Peer Evaluation

Computer science instructors are often concerned that when students program in pairs, some less motivated students may take advantage of their partner’s work. To alleviate this concern, we required the students to complete a peer evaluation for their partner at the end of each rotating cycle. The peer evaluation was administered via a web-based tool, which will also be explained in detail in Appendix B.

There are five quantitative questions and one qualitative question in the evaluation. The students rated their partner a scale from 0% to 20% for the first five quantitative questions, with a maximum additive score of 100%. The questions are listed below:

1. Did your partner read the lab assignment and preparatory materials before coming to the scheduled lab?
2. Did your partner do their fair share of the work?
3. Did your partner cooperatively follow the pair programming model (rotating roles of driver and navigator)?
4. Did your partner make contributions to the completion of the lab assignment?
5. Did your partner cooperate?
6. Write down any comments you have.

Each student's lab grade was multiplied by the value of their average peer evaluation score. For example, if a student had a lab score of 90, and he or she got a 70% in the evaluation, then his or her final lab grade is (90 * 70%) = 63.

3.2.3 Role Switching

In the lab, we periodically observed one student in the pair spending too much time driving. One important factor of pair programming is the two programmers are equal. They should make the same contributions to the task (Williams, Kessler et al. 2000). Therefore in Spring 2002, we add a new feature to the pairlearning lab – a kitchen timer. The timer was set for 20 or 30 minutes. When the timer dings, the student pairs were required to switch the driver and navigator roles. Thus, both students in a pair are assured an opportunity to be the driver.

We conducted a focus group with the students to obtain their feedback about pair learning. In the focus group, students extensively discussed the switching role: how much time the pairs should be given before being asked to rotate. They asserted that we needed to make the time period more flexible. One student said, "We paid the same tuition, we should get the same amount of time to get on the computer." Another said, "We will actively rotate among ourselves, the timer is disturbing." Another mentioned that there were some students that like to be the driver all the time and were not very nice about switching. These students would not rotate until the teaching assistant asked them to do so. After some discussions, they agreed the need to rotate should be mandatory, but the time can be flexible to be 20 or 30 minutes or depending on the pair themselves.

3.3 Step Two: Data Collection

To assess the efficacy of pair learning to the CS1 course, we collected a significant amount of data about the students. This data can be divided into two categories, background and performance data. First, we collected background data for each student. This background data includes: student name, gender, race, class, major, section number, and SAT-Math score from the NCSU registration and records office and from an online class archive. Because we were collecting background information on the students, we obtained permission to collect the data from the NCSU Institutional Review Board (IRB). The IRB declared the project exempt from surveillance, however students did voluntarily sign a release form allowing us to collect the data. Another bit of background information we collected was the students' programming background based on a programming assessment we administered, as discussed in Section 3.3.2.

The data in the performance category was collected from the students in lab or in class. This data can be further subdivided into sub-categories: (1) quantitative data from students' tests or programming assignment and (2) qualitative data achieved from lab observation or feedback from students and teaching assistant. The quantitative data include: score of lab1 through lab 10, two midterms and one final exam scores, three project assignment scores, final grade, peer evaluation feedback score (which was collected by a web-based program), attitude survey scores and programming assessment scores, which will be discussed in Section 3.3.1 and Section 3.3.2, respectively. The qualitative data included the comments students provided in their peer evaluations, teaching assistant feedback, lab observations about the students' behavior, and a focus group, discussed in Section 3.3.3.

3.3.1 Surveys

To assess student attitudes toward computer science, they were required to complete a questionnaire at both the beginning and end of the semester. The questionnaire was developed to measure attitudes towards computer programming and computer science in general. This instrument was derived from the Fennema-Sherman mathematics attitudes scales (Fennema and Sherman 1976), modified to reflect programming and computer science rather than mathematics (Wiebe 2001). The attitude survey is constituted of five parts:

- Confidence in Learning Computer Science;
- Attitude toward Success in Computer Science;
- Computer Science as a Female Domain;
- Usefulness of Computer Science Scale; and
- Effectance Motivation in Computer Science Scale.

There were 57 questions in the above five categories and an additional five questions asked about the students' age, gender, race and background information. For each of the 57 questions, there are five possible answers: a) strongly agree, b) agree, but with reservations, c) neutral, neither agree nor disagree, d) disagree, but with reservations and e) strongly disagree.

3.3.2 Programming Assessment

Due to the fact that many students (but not all) take a considerable amount of programming courses in high school, we administered a programming assessment for the students to complete during their first lab. Ms. Carol Miller, the instructor of the CS1 course, provided the programming assessment. It was focused on basic understanding of programming knowledge. This enabled us to gage know how much programming knowledge the students had before they enrolled in the CS1 course. We used the results of this instrument to evaluate the academic equivalence of the paired and the solo groups.

The programming assessment consisted of eight questions about the basic concepts of programming language. From the programming assessment score, we could see how much knowledge a student had before he/she began the CS1 course. Each teaching assistant would score the programming assessment for the students in their own lab.

3.3.3 Focus group

To obtain additional qualitative feedback from students and teaching assistants we also organized one student focus group and one teaching assistant focus group. In the student focus group, students shared their thoughts about pair learning and how it could be improved. Seven students participated in the student focus group: three white male, three foreign students, one African American. In the teaching assistant focus group, the teaching assistants shared their experiences about pair learning. There were four teaching assistants participated it.

Chapter 4 Experiment Findings

Our study was mostly concerned with the performance of beginning students. Therefore, we focused on analyzing the scores of the freshman and sophomores. In our analysis we thus excluded all the students who were in their junior or senior year or were graduate students. Additionally, we excluded the students who took the CS1 course for credit only or audited the class, surmising those students were not as motivated to excel as other students. This reduced our sample size to N = 69 in the solo section and N = 44 in the paired section.

4.1 Quantitative Findings

In our experiment, we measured the success / retention rate, examination scores and project scores of the experimental (pair learning) and control (solo) groups.

4.1.1 Success / Retention Rate

Historically, beginning Computer Science classes have a low success rate, often cited informally as about 50% nationally. We evaluated whether pair learning can help improve the success rate of the CS1.

Table 2 Success rate

Section	C and above	Below C	Success Rate
Pair	30	14	68.18%
Solo	31	38	44.93%

As shown in Table 2, we observed a 45% success rate in the solo section. In the paired section, 68% of the students completed the course successfully. A Chi-

Square test showed a significant difference between the two sections ($\chi^2 = 5.849$, $p < 0.016$). We have hypothesized that a larger percentage of students would finish the CS1 course with a grade of C or above in the paired than those in the solo section. These results indicated that the difference in success rate between the two sections is statistically significant. We, therefore, state that pair learning is a valuable education technique for helping beginning students succeed in their early computer science academic careers.

4.1.2 Performance on Examinations

We hypothesized that the students in the paired section would have higher examination scores than the solo students.

As shown in Table 3, on average, the paired section performed than the solo section in the examinations. When we calculated the statistics, we eliminated all the 0 scores; we did not count the students who did not take the exams. We can see the students in paired section outperformed than the solo students. But, we also want to analyze whether the difference is made because of the application of pair learning or for some other reason.

Table 3 Exam Scores

Exam	Paired Mean	Paired Standard Deviation	Solo Mean	Solo Standard Deviation
Midterm 1	78.7	11.8	73.4	13.8
Midterm 2	65.8	24.2	49.5	27.2
Final	74.1	16.5	67.2	18.4

As stated earlier, students chose the course sections without knowledge of this study. In theory, we hoped this would yield academically equivalent groups.

We performed statistical tests to assess the academic equivalence of the two sections, i.e. the students in both sections have similar aptitude. Our basis for comparison was their SAT-Math (SAT-M) scores and their scores on the programming assessment exam.

4.1.2.1 Academic Equivalence and the SAT-M

The students in the paired group had a mean SAT-M score of 662.10 while the solo section had a mean score of 625.43. The One-Way ANOVA (assuming equal variances between the two groups) revealed that the difference was statistically significant (*F* (1, 101) = 5.19, *p* < .018). An ANCOVA further revealed a correlation between SAT-M scores and exam scores (p < .0001 for all the three exams). This indicated that we needed to use the SAT-M as a covariate to guarantee the academic equivalence in the examinations. When using SAT-M as a covariate, an ANCOVA test showed there was a significant difference between the two sections in midterm 2 (*F*(1, 79) = 30.94, *p*< .036), but does not show a significant difference between sections with regards to midterm1 or final exam scores (*F*(1, 91) = 24.70, *p*<.335 for midterm 1; *F*(1, 71) = 25.96, *p*<.448 for final). Based on these results, we can state that the paired students did perform better on the exams, but the difference was only statistically significant for one of three exams. According to Williams (1999), there is a pair jelling period when the pair programmers began to work together. Pair learning was a new concept for our students. They also need to have a learning process for pairing with another student. Perhaps this explains why, the difference is in the first exam was not statistically significant. In the second exam, the students are more familiar with the pair learning protocol, so they began to be used to it. As a result, the difference in the second exam is statistically significant.

4.1.2.2 SAT-M as matching criteria

When we run the ANCOVA test for the exam scores with SAT-M as the covariate, we assumed that the exam scores were distributed normally. However, when we ran the Shapiro-Wilk test to test the normality of the three exams, the results ($p < .0001$ for midterm 1, $p < .0001$ for midterm 2, $p < .006$ for final exam) indicated that the exam score distributions were non-normal. The non-normal distribution suggests that non-parametric statistical tests would be more appropriate for evaluating the relationship of student learning skills and test scores, with SAT-M as a covariate. However, there is not a non-parametric equivalent for an ANCOVA. For that reason, an analysis based on post-hoc matched pairing based on SAT-M was performed. Within these groups, a Mann-Whitney U test could be used to examine differences in exam scores.

In order to do this we sorted the students in each section based on their SAT-M score. For the analysis, student results were then paired, one from the paired section and the other from the solo section. The pairs consisted of students having the same or similar SAT-M scores (the difference between the two scores is less than or equal to 10). For example, if a student with SAT-M of 750 in one section was picked out, then we chose a student whose SAT-M score is between 740 and 760 from the other section to form a pair with the previous student. If there are more than one student in the other group was qualified, then we randomly picked one.

We calculated the average scores and did statistical analysis in the groups we generated. Table 4 indicated that for the three exams, the paired section performed better than the solo section on average for the matched pairs. To test the statistical analysis, we ran the Mann-Whitney U test to assess whether the difference between these paired and solo sections was statistically significant. The results indicated (for midterm 1, *U* = 229.000, *z* = -2.345, *p* < .019; for midterm 2, *U* = 189.000, *z* = -2.042, *p* < .041; for final exam, *U* = 193.500, *z* = -1.140, *p* < .254) that the differences of midterm 1 and 2 between these two sections are statistically significant.

Table 4 Exam scores of matching pairs

	Paired Mean	Paired Std. Deviation	Solo Mean	Solo Std. Deviation
Midterm 1	78.296	11.4550	70.148	12.1424
Midterm 2	60.667	23.6251	43.208	30.2439
Final exam	70.500	17.1207	62.864	22.2867

4.1.2.3 Academic Equivalence and the Programming Assessment Score

We tested the statistical significance of the difference in programming assessment scores. The main purpose of this assessment was to examine the differences in the programming background the students had had before they took the CS1 course. Table 5 shows that the mean score of the programming assessment for paired section was higher than the solo section. We ran a t-test to test the statistical significance of the difference between these two sections. The results (*t* = 1.807, *p*< .080) showed that the difference is not statistically significant, assuming the variances are not equal. So from the programming background point of view, the two sections of students are academically equivalent to a statistically significant level.

Table 5 Programming Assessment Score

SECTION	N	Mean	Std. Deviation
Solo	58	1.7716	1.63988
Pair	23	2.6957	2.22455

Since the two sections of students have similar programming ability, and the distribution of the tests are non-normal, we ran a non-parametric test Mann-Whitney U test to assess whether the difference between the two sections are different. The results (U = 944.500, z = -2.201, p < .028 for midterm 1; U = 189.000, z = -2.042, p < .041 for midterm 2; U = 632.000, z = -1.693, p < .090 for final exam) indicated that the difference between the two sections in midterm 1 and midterm 2 are statistically significant, while this is not the case in the final exam.

Table 6 Summary of exam scores

			Statistically Significant?		
Examination	Mean Paired	Mean Solo	ANCOVA/ SAT-M	SAT-M Matching	Mann-Whitney U test
Midterm 1	**78.7**	73.4	No	Yes	Yes
Midterm 2	**65.8**	49.5	Yes	Yes	Yes
Final	**74.1**	67.2	No	No	No

We hypothesized that pair programmers would perform better on examinations. From the above experiment findings (see table 6), we can see that in general pair learning did help the beginning students to earn a better score on their examinations, although there is one exam for which the difference is not statistically significant.

4.1.3 Performance on Project assignments

As stated in Chapter 1, we hypothesized that the students in the paired section would do better in the project assignments than the students in solo section.

There were three projects in the Fall semester. Table 7 shows that, on average, students in the paired section performed better on two of the three project assignments. Again, when we performed calculations, we eliminated all the 0 scores; if a student did not hand in the projects, his or her score was not counted in the calculation.

Table 7 Programming Project Scores

Project number	Paired Mean	Paired Standard Deviation	Solo Mean	Solo Standard Deviation
Project 1	94.6	5.3	78.2	26.5
Project 2	86.3	19.7	68.7	33.7
Project 3	73.7	27.1	74.4	29.0

4.1.3.1 Academic Equivalence and the SAT-M

As we stated in 4.1.2.1, the difference between the two sections on the SAT-M score was statistically significant (*F* (1, 101) = 5.19, *p* < .018). Therefore, we did an ANCOVA test to find out whether there was a correlation between projects and the SAT-M scores. The results showed that for project 1 and project 2, the correlation between the project scores and SAT-M score is not statistically significant, while for project 3, the correlation was statistically significant (*F* (1, 69) = 7.186, *p* < .009). This indicated that we needed to use the SAT-M as a covariate for the project 3 to guarantee the academic equivalence in the project 3, while for the other two projects we only need to run them without considering covariance.

First, for the project 1 and project 2, we needed to test the normality of the grade distributions. So we ran the Shapiro-Wilk test to test the normality of the first two exams. The results (for project 1, statistic = .704, p < .0001; for project 2, statistic = .778, p < .0001) indicated that the distributions of project 1 and project 2 were not normal.

Second, since the distribution of the projects were not normal, we ran a non-parametric test Mann-Whitney U test, and the results (*U* = 882.500, *p* < .024 for project 1; *U* = 715.000, *p* < .160 for project 2) showed that the difference between the two sections on the project 1 is statistically significant; while it is not statistically significant for project 2.

Third, for the project 3, since the SAT-M affects the results of project 3, we ran an ANCOVA test using the SAT-M as the covariate, project 3 scores as the dependent variable, section number as the fixed factor. The results (*F (2, 69)* = 7.19, *p* < .0562) showed the difference between the two sections on project 3 after covariate by SAT-M is not statistically significant.

4.1.3.2 SAT-M as matching criteria for project 3

From the results in Chapter 4.1.3.1, we knew that SAT-M score did not significantly affect the project 1 and project 2 scores, so we only needed to run the ANCOVA test for the project 3 scores with SAT-M as the covariate. During this analysis, we assumed that the project 3 scores were distributed normally. However, when we ran the Shapiro-Wilk test to test the normality of the project 3, the results (statistic = .840, *p* < .0001) indicated that the project 3 score distribution is non-normal. As was done with examination scores in Chapter 4.1.2.2, a post-hoc matched pairing based on SAT-M was created between paired and solo groups. With these groups, a Mann-Whitney U test could be used to examine differences in exam scores.

Again, when we did the calculation, we eliminated all the 0s. Table 8 indicated that the project 3, paired section did not perform better than the solo section on average for the matching pairs. To test the statistical analysis, we ran the Mann-Whitney U test to assess whether the difference between these paired and solo sections are statistically significant. The results indicated that the difference between these two sections is not statistically significant.

Table 8 Project 3 scores of matched pairs

	Paired Mean	Paired Std. Deviation	Solo Mean	Solo Std. Deviation
Project 3	69.6522	30.18520	75.7717	27.00178

4.1.3.3 Academic Equivalence and the Programming Assessment Score

Same as we stated in Chapter 4.1.2.3, we also use the programming assessment score as the covariate for the three project scores. Since we knew from 4.1.2.3, in the programming background point of view, the two sections of students are not statistically significant different based on the programming assessment. The distribution of the tests was non-normal, so we ran a non-parametric test Mann-Whitney U test to assess whether the difference between the two sections was different. The results for project 1 and project 2 has been provided in Section

4.1.3.1, and the results for project 3 is U = 715.000, $p < .662$, which indicated that the difference between the two sections in project 3 is not statistically significant.

Table 9 Summaries of Project Scores

			Statistically Significant?		
Project	Mean Paired	Mean Solo	ANCOVA/ SAT-M	SAT-M Matching	Programming Assessment
Project 1	94.6	78.2	Yes	N/A	Yes
Project 2	86.3	68.7	No	N/A	No
Project 3	73.7	74.4	No	No	No

We hypothesized that paired students would perform better on programming assignments. From the above analysis, we see that pair learning did help students perform better on their programming assignments. However, due to the fact that the lower level students dropped out of the class in the solo section, which results in a higher average of the solo section, the difference between the paired students and solo students in only half of the projects is statistically significant.

4.1.4 Attitude

As stated in Chapter 1, we hypothesized that the students in the paired section would have more positive attitudes towards the computer science major than the solo section.

To evaluate the effect of the pair learning on the attitude of students, the students completed a questionnaire as discussed in Chapter 3.2.3. For our analysis, we gave a value to every answer for the questions. For example, answer a) strongly agree (value=5); answer b) agree, but with reservations (value = 4); answer c) neutral, neither agree nor disagree (value=3); answer d) disagree, but with reservations (value=2); answer e) strongly disagree (value=1). This is true for the positive questions.

There were also some negative questions, for example, “I’m not good at programming”. For those negative questions, we reversed the value of their answers. For example, if a student’s answer for the negative question is a) strongly agree, then the corresponding value is 1 instead of 5; if a student’s answer is b) agree, but with reservations, then the corresponding value is 2 instead of 3 and so on.

We desired to summarize the students’ attitudes by compiling student results into subscales. Dr. E. N. Wiebe (personal communication, Mar. 26, 2002) provided the Cronbach Coefficient Alpha test data (see table 10) to measure the internal consistency of each of these five subscales in the attitude survey. The assumption was all of the questions within a subscale (for example, the Confidence subscale) measured the same attribute and therefore individuals should answer all of the questions within the subscale similarly. Cronbach Coefficient Alpha measures this level of consistency. If the value on this test is over 0.8, it is considered a valid instrument.

Table 10 Internal Consistency Value

Subscales Test Value	**Cronbach Coefficient Alpha**
Confidence	0.91
Attitude toward Success	0.86
Female Domain	0.83
Usefulness	0.91
Effectance Motivation	0.90

Because we proved internal consistency, we could add the students' answers for each question in a subscale (see Table 11 and 12). Table 11 displays the values the students did at the beginning of the semester; Table 12 displays the values at the end of the semester. We ran a non-parametric analysis of variance (Mann-Whitney U). The test indicated that there was no significant difference in any of these categories between the paired and the solo lecture sections ($p > .217$ for all five categories) either at the beginning or end of the semester. From these results, we could not conclude that pair learning in the CS1 course helps the students have a more positive attitude towards computer science solely. However, we can see the trend that all the students had higher confidence at the end of the semester than the beginning of the semester.

Table 11 Attitude survey scores at the beginning of the semester

Section	Paired Sample Number (N)	Paired Mean	Paired Std. Deviation	Solo Sample Number (N)	Solo Mean	Solo Std. Deviation
Confidence	66	26.98	10.001	98	29.52	9.497
Success	65	21.65	7.688	97	20.92	6.819
Female	66	15.80	5.393	98	14.89	5.197
Usefulness	66	23.20	7.814	96	23.82	8.823
Effectance	66	28.62	9.566	95	29.12	8.606

Table 12 Attitude survey scores at the end of the semester

Section	Paired Sample	Paired Mean	Paired Std. Deviation	Solo Sample	Solo Mean	Solo Std. Deviation

	Number (N)			Number (N)		
Confidence	50	29.22	13.140	54	31.93	12.700
Success	50	23.96	8.635	54	23.35	8.414
Female	50	17.34	6.699	53	15.62	6.464
Usefulness	50	28.46	12.551	53	27.51	10.392
Effectance	50	29.90	10.531	53	31.11	10.994

We also examined the NCSU course evaluations related to the students' attitude. The only data available on the course evaluation was a mean score, so no statistical evaluation could be performed. On the course evaluation, a 1 is an unfavorable score and a 5 is a very favorable score. As shown in Table 13, students did feel more favorable in the paired section, though the instructor, material and evaluation artifacts were identical:

Table 13 Course Evaluations

	Paired Mean	**Solo Mean**
Course Effectiveness	3.97	3.58
Instructor Effectiveness	4.20	3.69
Classroom is Instructive to Learning	4.26	4.26

We hypothesized that pair programming would cause the students to have more positive attitudes about computer science. From the above analysis, we cannot support this hypothesis.

4.2 Qualitative Findings

To better study the pair learning technique in the labs, we observed paired and solo labs sessions. Following are the observation results we collected during the

Fall 2001 and Spring 2002 semesters. In the Fall 2001 semester, we observed students not rotating the roles of driver and navigator when they programmed. As a result, there were students who did not have an opportunity to be the driver. To avoid this, in Spring 2002 semester, each lab was equipped with a kitchen timer. Teaching assistants set the timer for 20 or 30 minutes. When the timer went off, students were required to rotate roles.

4.2.1 Students' Behaviors during Pair Learning

In the paired section, we observed a great deal of discussions among the students. Student pairs brain stormed with each other to solve their programming assignments. Since they tended to figure things out amongst the two in the pair, they rarely asked the teaching assistant questions. If they did, the questions were mostly logistical in nature; for example, how they can improve more upon their programs instead of operational questions like how to create a new directory. Most of the pairs switched the "driver" and "navigator" roles when the timer went off or when they came to a certain point shortly after the timer went off. Few pairs were reluctant to change. But the students showed increasing willingness to switch as the semester progressed (Williams, Wiebe et al. 2002).

In the solo section, the labs were fairly quiet. Students had questions on a more frequent basis, so they raised their hands and waited for the teaching assistant to help them. The maximum time a student waited before the teaching assistant came to help him was thirty minutes. During this time the waiting students made very little progress, if any. Some students even gave up trying to get the teaching assistant's attention and turned to their neighbors. When the teaching assistant helped the students, they tended to take over the students' keyboard. One student who withdrew from the CS1 course the year before and was in the paired lab in the Spring semester shared his experiences in both semesters, "When I had the CS1 course last year in the solo lab I had a very simple syntax error, but I just could not figure it out at that time. So I asked the teaching assistant. When I finally got hold of him after waiting for a while, he seemed reluctant to sit down to read the code line by line with me, and it turned out I need to solve it myself. While in the paired lab, my partner sat with me and pointed out the syntax error when I just made it."

4.2.2 Teaching assistant's workload

As previously discussed, in the paired section, students were not as reliant upon the teaching assistant for technical advice. On average, the teaching assistant spent very little time answering questions, generally less than 5 minutes each. Sometimes, they even had time to do their own work. Whereas in the solo lab, we observed much hand waving to get attention of the teaching assistant. The time the teaching assistant spent to answer a question is was longer, say from 5 minutes to 20 minutes. One teaching assistant who had been teaching both the paired and solo sections said, "I got fewer questions in the paired lab and the questions the paired lab asked are more reasonable. In the solo section, there was a student asking how to create a directory in the 7th lab, which he should have known in the first lab."

As far as grading is concerned, it is obvious that the grading load is halved due to one pair submit one copy of homework.

We administered a questionnaire to the teaching assistants to obtain feedback on their views of pair learning and their observations about students' behavior. There were ten teaching assistants, and five out of the ten replied. Among the five teaching assistants (TA), two were responsible for paired labs, whereas three are responsible for solo labs. Moreover, one of the two paired lab TAs taught both the paired lab and solo lab. There seven questions, and the first four is for both paired and solo section TAs, and the last three is for the paired section TA only. The questions in the questionnaire are listed below (Q1: question 1), and the answers from the teaching assistants are after the questions (A1: Answer to the first question). Since the population of the respondents is too small, we did not do the statistical analysis.

Q1: How many students were in your lab?

A1: On average, there were around 20 students.

Q2: What percentage of lab time did you spend answering students' questions?

A2: The two paired lab TAs spent 10% and 25% of the lab time to answer students' questions respectively; while the solo labs TAs' answer were 90%-95%, (essentially the whole lab time), and 75% respectively.

Q3: Give several examples (at least 2) of the questions you were asked.

A3: The solo students asked like, “In our project, we're supposed to do such and such, I don't really understand how to do that, can you help? ”; while paired students asked questions like, “What is a better way to code 'x' method? ”.

Q4: If you were a teaching assistant again, would you want to teach a paired or solo lab?

A4: Most of the TAs said paired lab was interesting.

Question 5 – 7 is answered by paired section TA only:

Q5: Based on your previous experiences as a teaching assistant or as CSC116 student, if you were a paired lab teaching assistant, could you please state at least three differences you can think of between the solo labs?

Q6: On a scale from 1 (very receptive) to 5 (very resistant), rate how receptive your students were towards pair programming at the beginning of the semester.

A6: One of the TAs enumerated the difference between the paired lab and the solo lab, "The main differences are the effects on the TA position. The grading is much easier in the paired lab. The code is clearer in the paired lab having been proofed by the other student while writing. The students in the paired lab thought the labs were easier than the non-paired students thought. The time spent in lab was roughly only 3/5 the time spent in my non-paired sections. The extra work given by the paired students was obvious. MOST IMPORTANTLY: The extra time gained by not having to answer as many questions allowed me more time to help the better students do even better. It also allowed me to get to know the other students better. In non-paired sections there are always a few that eat up your time asking stupid questions. That was not the case in the paired lab. ALSO IMPORTANTLY: The effects of having a lab where communication is required opened up the students to talk to me more as well as their partners. They were much more open than non-paired students in talking to me about problems, concerns, etc." The other TA also said about his concerns, "In non-paired labs, students don't get help from their peers, which is a minus. In paired labs sometimes students feel like they are dead weight if their partner is very experienced so they might not want to work with someone else".

Q7: On a scale from 1 (very receptive) to 5 (very resistant), rate how receptive your students were towards pair programming at the end of the semester.

A7: both of the TAs in paired lab answered the students grew up on the pair learning.

So from our analysis above, the teaching assistant did have lower workload in grading and answering questions.

Chapter 5 Conclusion and Future Work

A formal experiment was performed in a CS1 course, “Introduction to Java”, in North Carolina State University. The purpose of this research was to assess whether pair learning can help the entry level students learn better in the computer science courses, help them more confident in the courses, and as a result, more students will succeed in computer science. The following hypothesis was tested:

- More students that have participated in pair learning in CS1 will succeed in the class by completing the class with a grade of C or better.
- Students’ participation in pair learning in CS1 will lead to better performance in examinations (exams are completed solo by all students) in that class.
- Students’ participation in pair learning in CS1 will lead to better performance on course projects in that class.
- Students’ participation in pair learning in CS1 will lead to a more positive attitude toward the course and toward Computer Science in general.
- Students’ participation in pair learning lead to a lower workload for course assistant.

199 students participated in this study, 112 in solo section and 87 in the paired section. Since we focused primarily on “beginning students”, we eliminated the graduate students, junior and senior students. As a result, 69 students in the solo section and 44 in the paired section comprised our experiment samples.

We analyzed the students' three exam scores, three project scores, success/retention rate, programming assessment scores, questionnaire values to validate the hypothesis.

We hypothesized that more students who have participated in pair learning in CS1 will succeed in the class by completing the class with a grade of C or better. We found that pair learning did help students succeed by completing the class with a grade of C or better.

We hypothesized that paired students would perform better in the examinations. We found that, on average, the paired section students performed better than the solo section students, although the difference in one of the exam is not statistically significant. Further to guarantee the academic equivalence, we considered student SAT-M and programming assessment scores and found out that SAT-M score did affect the performance of students. We matched the students into pairs according to similar SAT-M scores and proved that the differences of midterm 1 and midterm 2 are statistically significant, but this was not the case for final exam. From the population of students that participated in the final exam, we observed more students dropped out in solo section than that in the paired section, so this resulted in the lower part of the solo section is gone. This phenomenon could explain why the difference of final exam between the two sections is not statistically significant while the other two are. So in general, we observed that pair learning did help students perform better in exams, although the difference in one of the three exams is not statistically significant.

We hypothesized that the experimental section would perform better in the programming assignments. We found that, on average, paired section students performed better than solo section on project 1 and project 2. In project 3, the solo section students did slightly better than the paired section. The reason we raised above can also interpret this phenomenon. So in general, we can see that pair learning did help students perform better in the projects.

We hypothesized that pair learning could help students be more confident and positive toward computer science. We found that the result of the questionnaire demonstrated an improving trend for the students, although the difference is not statistically significant. The attitude change is a long-run term, so we could not change it in one course. We expect that students' attitudes will be improved towards computer science after they took several paired section class.

We also hypothesized that pair learning can alleviate teaching assistants' working load in both grading and answering questions. We observed the teaching assistant's burden was indeed lowered: as evidenced by the need for less grading work and more time in labs.

Paired students demonstrated higher-order thinking skills than the students when compared with the students who worked alone. They were more likely to think far beyond the programming assignment to applying their knowledge in more challenging programming contexts. Additionally, paired students collaborated more extensively with each other; the collaboration skill will benefit them in the future when they work, since collaboration is an important skill for programmers in industry (Demarco and Lister 1987).

5.1 Future work

We had also hoped to study the effect of pair learning on the female students and African-American students. However, there were far too few women (12 in solo section, 4 in paired section) and African American (8 in solo section, 6 in paired section) to allow statistical evaluation. We hope to accumulate enough results of women over several semesters to yield statistically significant results.

The pair learning technique may also be improved by learning more about prudent pair assignments. Both in the student and teaching assistant focus groups, we heard that the partner played an important role in the communication, directly affecting the results of the learning process. Some students said their partners were great and he/she learned a great deal from them. Others may complain that it was difficult to collaborate with their partners. Having a good partner is very important in pair learning. Students would prefer to pick their own partners.   But when they just came to university, they hardly know anyone. So they needed to be “randomly” assigned. Future research should address this “random” assignment to determine if a pattern for best matching each student into pairs, for example based on their background equivalences or their personality type.

Finally, it would be interesting to analyze the interactions of the student pair learners from a psychological perspective to see how they really communicate with each other. As a result, we can further find out how pair learning can benefit the students psychologically.

Appendix A Web-based Tool Support

As we can saw from previous chapters, there were two web-based applications written to support pair learning and this research. One tool was used by the teaching assistants to automatically assign the pairs each week; the other tool was used by the students to do the peer evaluation.

A.1 Pair assignment program

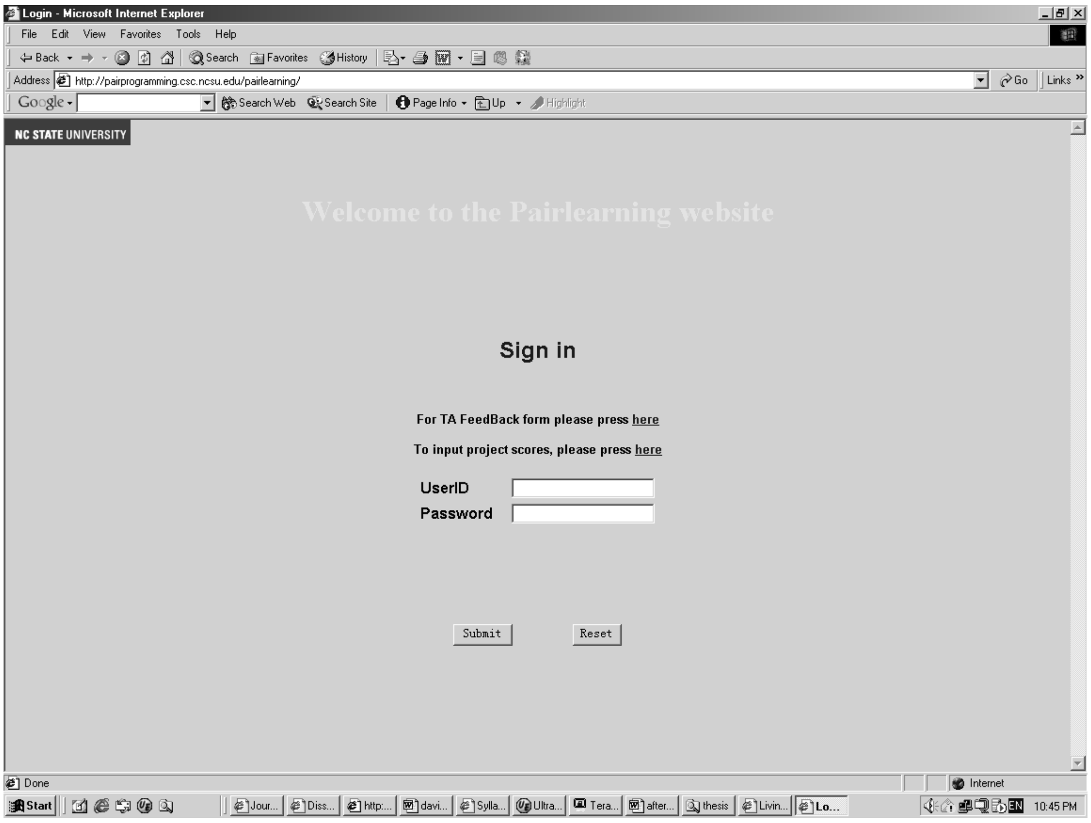
A.1.1 Pair assignment algorithm implementation

As we stated before, the students were assigned partners rather than allowing them to choose their partner. The partner assignment program was used to make these assignments. This application is implemented using Java Server Pages (JSP) with MS SQL Server database.

The purpose of this algorithm is to avoid the students to be with the same partner for too long, i.e. students should change partner every cycle. The students' names are input to the database, ordered alphabetically. For the cycle 1, the odd number students are assigned to their next neighbor, i.e. student (2 * i – 1) is paired with student (2 * i), where i is a positive integer. If the total number of students is odd, then the last student will pair with the first two students to have a 3-member group. For the cycle 2, the students are assigned to pair with the student next to his neighbor, i.e. student i is paired with student (i + 2). If the total number of students is odd, then the left student will be assigned to the first group. For the cycle 3, the students are assigned to pair with the student next to his neighbor, i.e. student i is paired with student (i + 3). If the total number of students is odd, the left student would be assigned to the second group. And so on. In general, for the cycle N, the student i is paired with student (i + N). If the total number of students is odd, the left student will be assigned to the (N-1)th group. In this way we can fulfill our purpose that the students can pairs as many as the cycle number.

A.1.2 Program flow

When a teaching assistant comes to this website, he/she needs to sign on first (as in figure 7.1). If the username and password are not right, he/she can not access the web page. Those given access are then directed to another page asking him or her to choose from the lab sections he/she is responsible for (as in figure 7.2). After that, an assignment of the students in that section shows on the web (as in figure 7.3). Then the teaching assistant can assign the student according to the contents on the web page.


Login - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://pairprogramming.csc.ncsu.edu/pairlearning/
Go Links
Search Web Search Site Page Info Up Highlight
NC STATE UNIVERSITY
Submit
Reset
Done
Internet
Start
10:45 PM

Welcome to the Pairlearning website

Sign in

For TA FeedBack form please press here

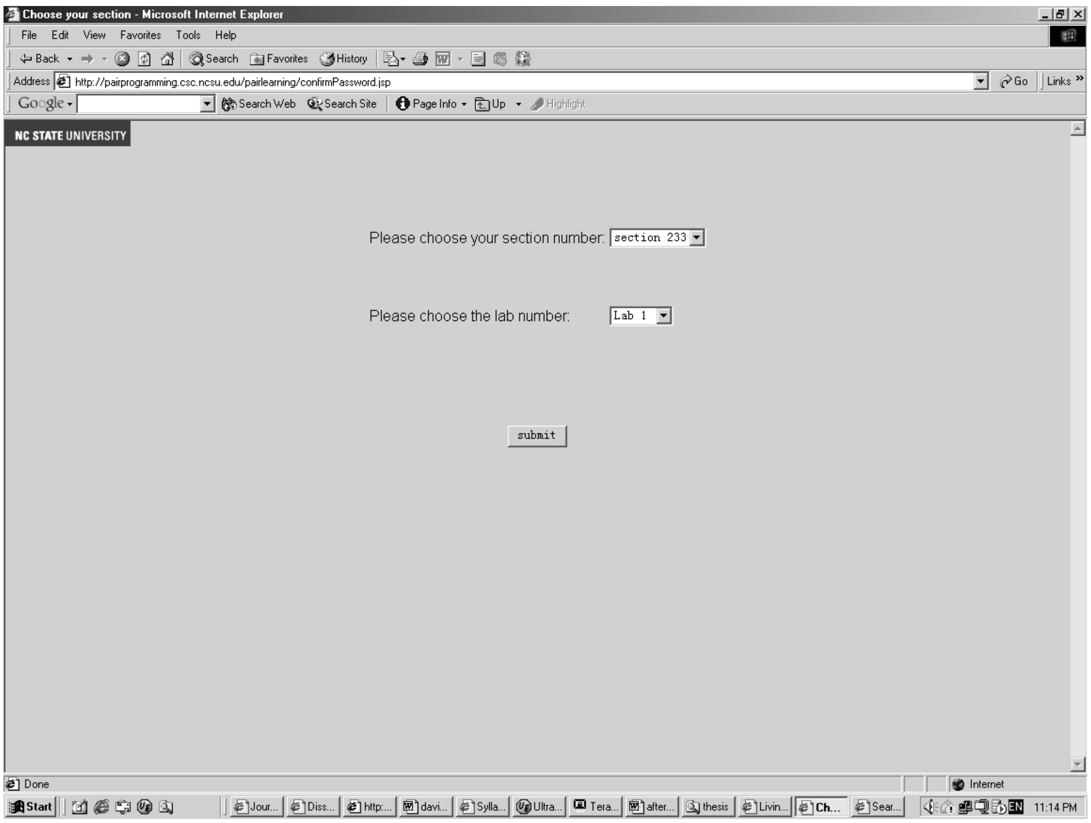
To input project scores, please press here

UserID

Password

Figure 2 Log in page

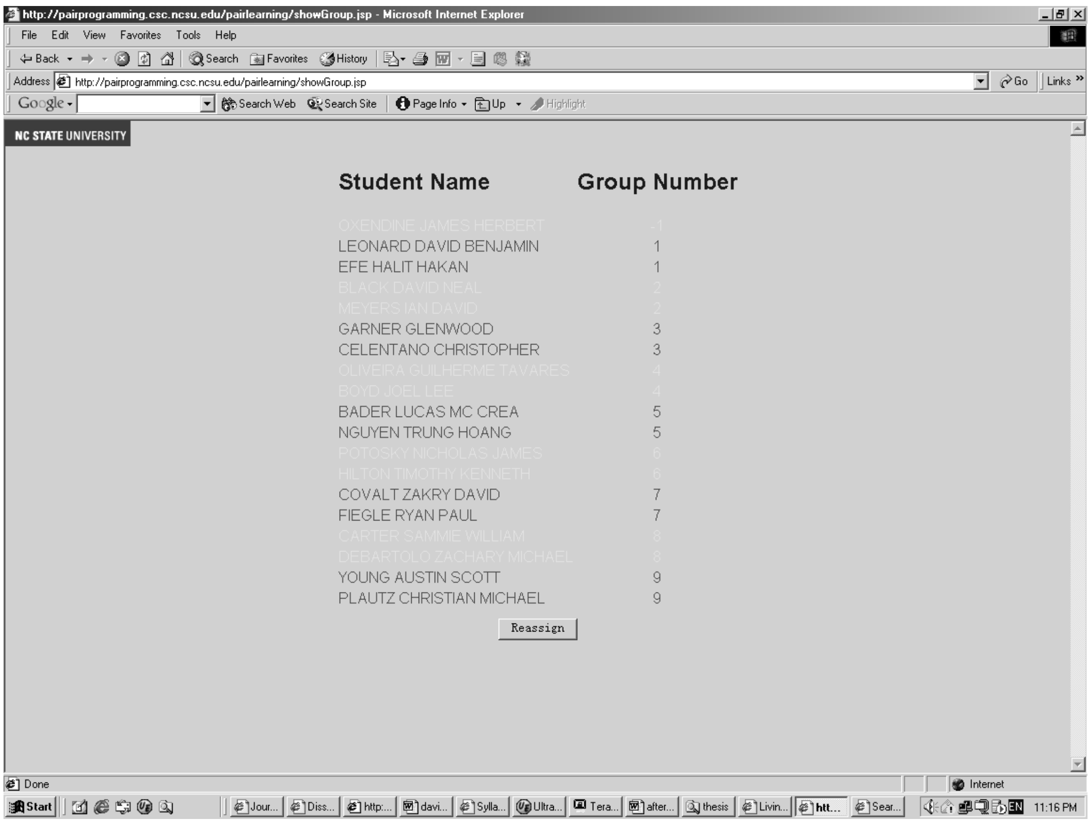
If a student is absent from a lab or for any reason, the teaching assistant can also reassign the student pairs and record any changes (as in figure 7.4). If a student was reassigned a partner, the teaching assistant should record the new group number on the website, and the data will be updated in the database. Updating the revised assignment is important, because the actual, accurate pairings are necessary as input into the peer evaluation tool, which is discussed below.


Choose your section - Microsoft Internet Explorer
http://pairprogramming.csc.ncsu.edu/pairlearning/confirmPassword.jsp
NC STATE UNIVERSITY
submit

Please choose your section number: section 233

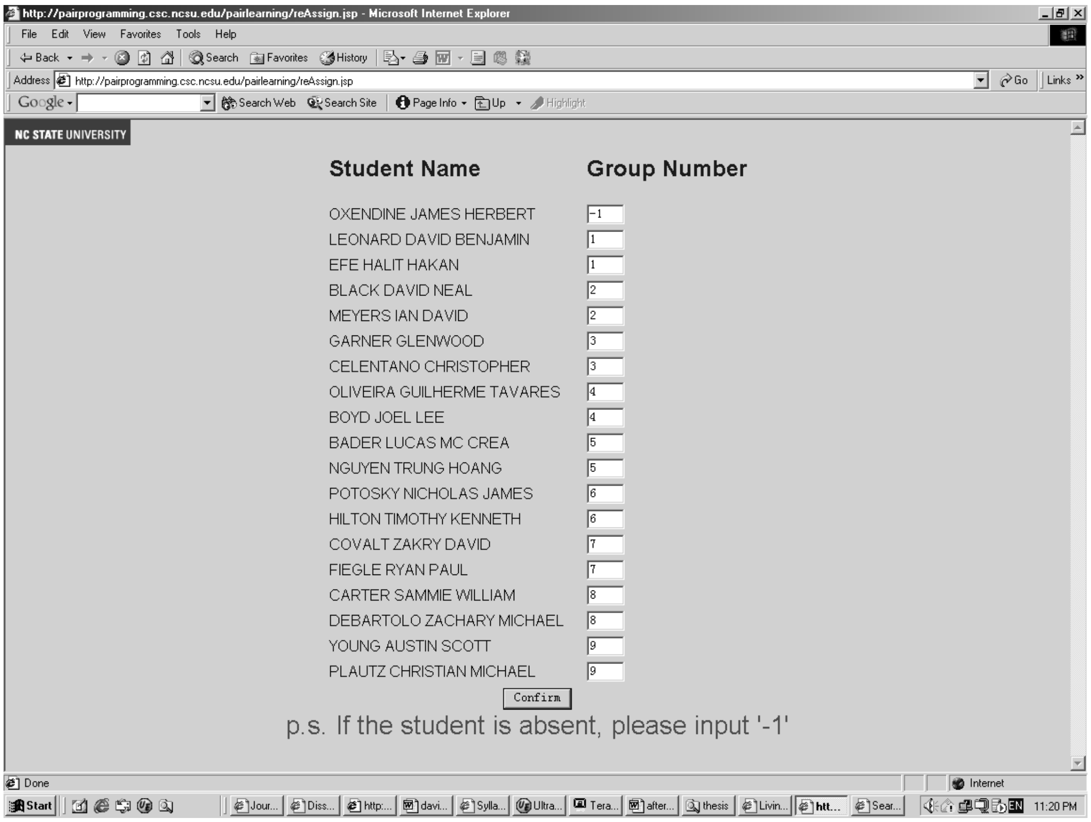
Please choose the lab number: Lab 1

Figure 3 choose lab section


http://pairprogramming.csc.ncsu.edu/pairlearning/showGroup.jsp - Microsoft Internet Explorer
NC STATE UNIVERSITY
Reassign

Student Name	Group Number
OXENDINE JAMES HERBERT	-1
LEONARD DAVID BENJAMIN	1
EFE HALIT HAKAN	1
BLACK DAVID NEAL	2
MEYERS IAN DAVID	2
GARNER GLENWOOD	3
CELENTANO CHRISTOPHER	3
OLIVEIRA GUILHERME TAVARES	4
BOYD JOEL LEE	4
BADER LUCAS MC CREA	5
NGUYEN TRUNG HOANG	5
POTOSKY NICHOLAS JAMES	6
HILTON TIMOTHY KENNETH	6
COVALT ZAKRY DAVID	7
FIEGLE RYAN PAUL	7
CARTER SAMMIE WILLIAM	8
DEBARTOLO ZACHARY MICHAEL	8
YOUNG AUSTIN SCOTT	9
PLAUTZ CHRISTIAN MICHAEL	9

Figure 4 Student assignments


http://pairprogramming.csc.ncsu.edu/pairlearning/reAssign.jsp - Microsoft Internet Explorer
NC STATE UNIVERSITY
Confirm

Student Name	Group Number
OXENDINE JAMES HERBERT	-1
LEONARD DAVID BENJAMIN	1
EFE HALIT HAKAN	1
BLACK DAVID NEAL	2
MEYERS IAN DAVID	2
GARNER GLENWOOD	3
CELENTANO CHRISTOPHER	3
OLIVEIRA GUILHERME TAVARES	4
BOYD JOEL LEE	4
BADER LUCAS MC CREA	5
NGUYEN TRUNG HOANG	5
POTOSKY NICHOLAS JAMES	6
HILTON TIMOTHY KENNETH	6
COVALT ZAKRY DAVID	7
FIEGLE RYAN PAUL	7
CARTER SAMMIE WILLIAM	8
DEBARTOLO ZACHARY MICHAEL	8
YOUNG AUSTIN SCOTT	9
PLAUTZ CHRISTIAN MICHAEL	9

p.s. If the student is absent, please input '-1'

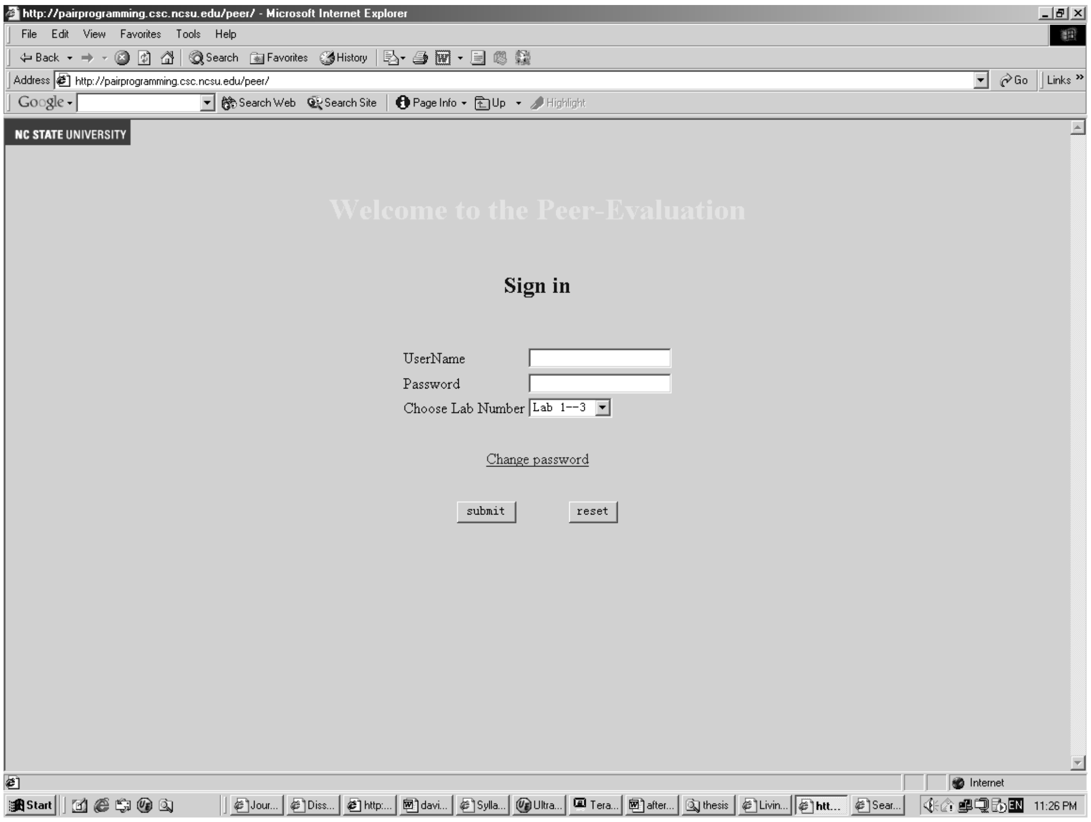
Figure 5 Change student pairs

A.2 Peer Evaluation Program

A.2.1 Program design

The purpose of this program is to assure the students all actively make contributions in the group to avoid some students taking advantage of their partner's work. It is also implemented using JSP and MS SQL Server database.

When a student comes to the website, he/she will be asked to log in first (as shown in figure 7.5). He/she can also choose which lab he/she is going to do the evaluation for. If the password and username are right, he/she will be directed to the next page to answer the questions (as shown in figure 7.6). Each question will be scaled from 0 – 20. If he/she answered a wrong character or a number out of the boundary, then a JavaScript warning box will be pop out.


http://pairprogramming.csc.ncsu.edu/peer/ - Microsoft Internet Explorer
NC STATE UNIVERSITY
submit
reset
Change password

Welcome to the Peer-Evaluation

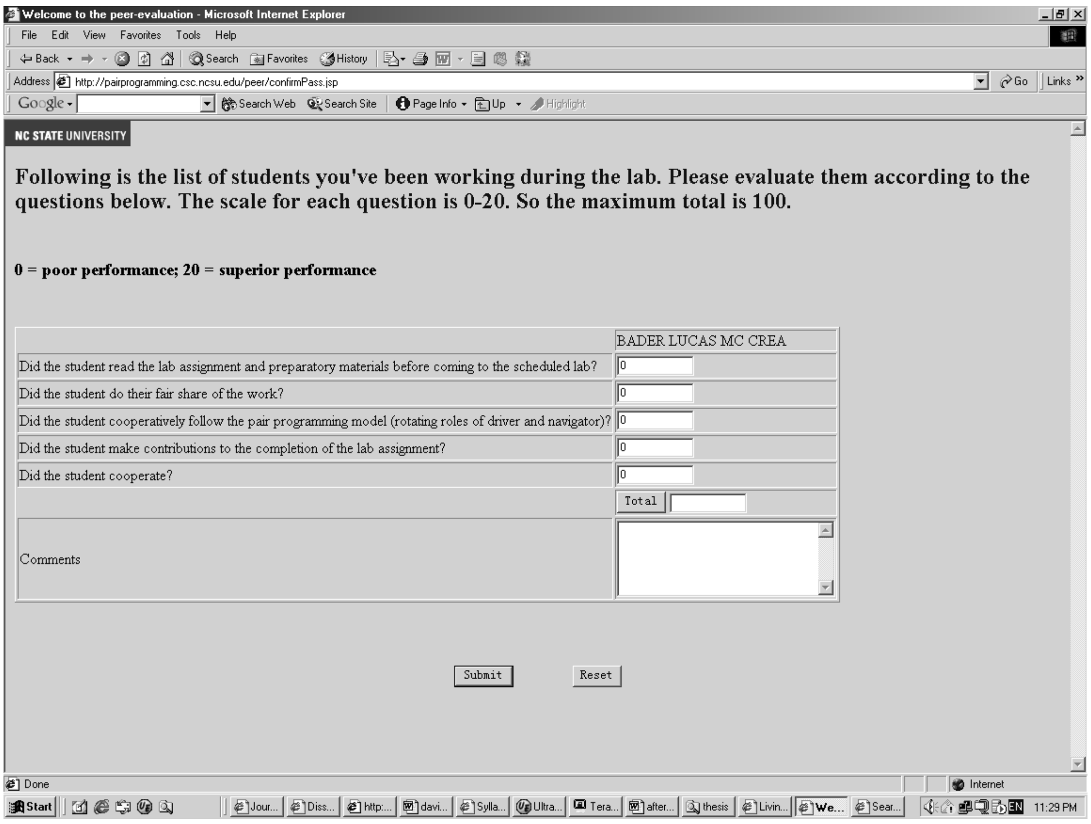
Sign in

UserName

Password

Choose Lab Number Lab 1--3

Figure 6 Peer Evaluation Log in Page



Welcome to the peer-evaluation - Microsoft Internet Explorer

http://pairprogramming.csc.ncsu.edu/peer/confirmPass.jsp

NC STATE UNIVERSITY

Following is the list of students you've been working during the lab. Please evaluate them according to the questions below. The scale for each question is 0-20. So the maximum total is 100.

0 = poor performance; 20 = superior performance

	BADER LUCAS MC CREA
Did the student read the lab assignment and preparatory materials before coming to the scheduled lab?	0
Did the student do their fair share of the work?	0
Did the student cooperatively follow the pair programming model (rotating roles of driver and navigator)?	0
Did the student make contributions to the completion of the lab assignment?	0
Did the student cooperate?	0
	Total
Comments	

Submit Reset

Figure 7 Peer Evaluation Page

Appendix B Programming Assessment

We're attempting to assess your programming background and skills. Answer as many question as you can, even if you have little or no programming experience. If you don't know how to answer a question you can simply write 'do not know'.

Before you begin, please provide your name and indicate what programming language you will be using:

Name:____

Programming Language:
____

Arithmetic

1. Use integer arithmetic to evaluate the following expressions. Use standard integer arithmetic as implemented in common programming languages (e.g. C, C++, Basic, Java)

3 + 5 * 2 = ______

4 * 3 + 6 = ______

7 / 2 + 4 = ______

13 / 4 / 2 = ______

2. If / and + have the **same precedence**, what is the value of

3 + 1 / 4 / 2 + 5 = ________________

If / has **higher precedence** than +, what is the value of that same expression?

3 + 1 / 4 / 2 + 5 = ________________

Selection

3. Assume an integer variable named **temperature** has been declared and assigned a value.
Write a statement that displays "Water" to the console if the value of **temperature** is between 32 and 212 (inclusive).

Iteration (loops)

4. Write a loop that sets a variable named **sum** to the sum of the even numbers between 1 and 99. Assume that **sum** has been declared and initialized to zero.

5. Use nested loops to produce the following console output:

```
*
**
***
****
```

Arrays

6. Assume an integer array named **intArray** has been initialized to hold 10 values. Write a code segment that displays (to the console) the *smallest value* stored in the array.

Encapsulation

7.Some programming languages use **class**es. Explain what a **class** is and provide a short program example.

8. **Functions** are commonly used in programming languages. Explain what a **function** is and provide a short example. If your programming experience has been in Java or C++, explain **methods** instead, and provide a method for the **class** you just provided.

Appendix C Computer Science / Programming Interests Questionnaire

Computer Science / Programming Interests Questionnaire (Wiebe 2001)

Directions

Enter your student ID number onto the answer sheet. Please note that your answers will be kept confidential.

On the following pages are a series of statements.

1. Read each statement.
2. Think of the extent to which you agree or disagree with each statement
3. Mark your response on the answer sheet

Please remember:

- There are no right or wrong answers. Don't be afraid to put down what you really think.
- Don't spend a lot of time on any one item. Move quickly!
- Complete all of the items.

Respond to the following questions on the answer sheet, using the following scale:

a) strongly agree
b) agree, but with reservations
c) neutral, neither agree nor disagree
d) disagree, but with reservations
e) strongly disagree

---

1. I plan to major in computer science.
2. Generally I have felt secure about attempting computer programming problems.
3. I am sure I could do advanced work in computer science.
4. I am sure that I can learn programming.
5. I think I could handle more difficult programming problems.
6. I can get good grades in computer science.
7. I have a lot of self-confidence when it comes to programming.
8. I'm no good at programming.
9. I don't think I could do advanced computer science.
10. I'm not the type to do well in computer programming.
11. For some reason even though I work hard at it, programming seems unusually hard for me.

12. Most subjects I can handle O.K., but I have a knack for flubbing up programming problems.
13. Computer science has been my worst subject.
14. It would make me happy to be recognized as an excellent student in computer science.
15. I'd be proud to be the outstanding student in computer science.
16. I'd be happy to get top grades in computer science.
17. It would be really great to win a prize in computer science.
18. Being first in a programming competition would make me pleased.
19. Being regarded as smart in computer science would be a great thing.
20. Winning a prize in computer science would make me feel unpleasantly conspicuous.
21. People would think I was some kind of a nerd if I got A's in computer science.
22. If I had good grades in computer science, I would try to hide it.
23. If I got the highest grade in computer science I'd prefer no one knew.
24. It would make people like me less if I were a really good computer science student.
25. I don't like people to think I'm smart in computer science.
26. Females are as good as males at programming.
27. Studying computer science is just as appropriate for women as for men.
28. I would trust a woman just as much as I would trust a man to figure out important programming problems.
29. Women certainly are logical enough to do well in computer science.
30. It's hard to believe a female could be a genius in computer science.
31. It makes sense that there are more men than women in computer science.
32. I would have more faith in the answer for a programming problem solved by a man than a woman.
33. Women who enjoy studying computer science are a bit peculiar.
34. I'll need programming for my future work.
35. I study programming because I know how useful it is.
36. Knowing programming will help me earn a living.
37. Computer science is a worthwhile and necessary subject.
38. I'll need a firm mastery of programming for my future work.
39. I will use programming in many ways throughout my life.

40. Programming is of no relevance to my life.
41. Programming will not be important to me in my life's work.
42. I see computer science as a subject I will rarely use in my daily life.
43. Taking computer science courses is a waste of time.
44. In terms of my adult life it is not important for me to do well in computer science in college.
45. I expect to have little use for programming when I get out of school.
46. I like writing computer programs.
47. Programming is enjoyable and stimulating to me.
48. When a programming problem arises that I can't immediately solve, I stick with it until I have the solution.
49. Once I start trying to work on a program, I find it hard to stop.
50. When a question is left unanswered in computer science class, I continue to think about it afterward.
51. I am challenged by programming problems I can't understand immediately.
52. Figuring out programming problems does not appeal to me.
53. The challenge of programming problems does not appeal to me.
54. Programming boring.
55. I don't understand how some people can spend so such time on writing programs and seem to enjoy it.
56. I would rather have someone give me the solution to a difficult programming problem than to have to work it out for myself.
57. I do as little work in computer science courses as possible.

Please answer the following questions about yourself on the answer sheet:

58. Age:
    a) 18 years old or younger
    b) 19 years old
    c) 20 years old
    d) 21 - 30 years old
    e) 31 years old or older

59. Gender:
    a) Female
    b) Male

60. Classification (Grade Level):
    a) Freshman

b) Sophomore
c) Junior
d) Senior
e) Post-undergraduate or Graduate

61. Number of computer science or programming courses you have previously taken in high school or college:
a) None
b) One course
c) Two courses
d) Three or four courses
e) more than four courses

62. Ethnicity/Race:
a) White (non-Hispanic)
b) Black (non-Hispanic)
c) Asian American
d) Hispanic American
e) Others

Thank you for your time! Please let us know if you have any questions about this questionnaire or the study we are conducting. Questions or concerns can either be directed to the instructor of this course or the project director, Dr. Laurie Williams, Dept. of Computer Science, williams@csc.ncsu.edu, 513-4151

REFERENCES

Angelo, T. A. and K. P. Cross (1993). Classroom assessment techniques - a handbook for college teachers. New York, Jossey-Bass Publishers.

Beck, K. (2000). Extreme programming explained: embrace change, Addison-Wesley.

Bevan, J., L. Werner, et al. (2002). Guidelines for the use of pair programming in a freshman programming class. Fifteenth Conference on Software Engineering Education and Training, Covington, Kentucky, IEEE Computer Society Press.

Bonwell, C. C. (1996). Using Active Learning in College Classes: A Range of Options for Faculty.

Demarco, T. and T. Lister (1987). Peopleware. New York, Dorset House Publishers.

Felder, R. M. (1995). "A Longitudinal Study of Engineering Student Performance and Retention. IV. Instructional Methods and Student Responses to Them." Journal of Engineering Education **84**(4): 361-367.

Felder, R. M., G. N. Felder, et al. (1998). "A Longitudinal Study of Engineering Student Performance and Retention. V. Comparisons with Traditionally-Taught Students." Journal of Engineering Education **87**(4): 469-480.

Fennema, E. and J. A. Sherman (1976). "Fennema-Sherman mathematics attitudes scales." JSAS: Catalog of Selected Documents in Psychology **6**(31).

Haller, C. R., V. J. Gallagher, et al. (2000). "Dynamics of Peer Education in Cooperative Learning Workgroups." Journal of Engineering Education **89**(3): 285-293.

Humphrey, W. S. (1995). A Discipline for Software Engineering.

Jefferies, R. (1999). Pair programming.

Lemos, R. S. (1979). "An implementation of structured walk-throughs in teaching COBOL Programming." Communications of the ACM **22**: 335-340.

Leuf, B. and W. Cunningham (1999). Pair Programming. **2002**.

Lorenzen, M. (2002). Active Learning and Library Instruction. **2002**.

MacGregor, K. S. (1998). "Computer programming instruction: effects of collaboration and structured design mileposts." Journal of Research on Computing in Education **21**: 155-164.

McDowell, C., L. Werner, et al. (2002). The effects of pair programming on performance in an introductory programming course. Conference of the Special Interest Group of Computer Science Educators, Northern Kentucy, ACM Press.

Nosek, J. T. (1998). "The Case for Collaborative Porgramming." Communications of the ACM **41**(3): 105-108.

Priebe, R. L. (1997). The effects of cooperative learning on content comprehension in a second-semester university computer science course. Science Education. Austin, University of Texas at Austin.

Ruhl, K. L., C. A. Hughes, et al. (1987). Using the pause procedure to enhance lecture recall. Teacher Education and Special Education. **10**.

Slavin, R. E. (1990). Cooperative Learning Theory, Research, and Practice.

Wiebe, E. N. (2001). Computer Science / Programming Interests Questionnaire.

Williams, L. A. (2000). The Collaborative Software Process. Computer Science. Salt Lake City, The University of Utah**:** 186.

Williams, L. A. and R. R. Kessler (2002). Pair Programming Illuminated. Boston, Addison-Wesley.

Williams, L. A., R. R. Kessler, et al. (2000). "Strengthening the Case for Pair Programming." IEEE Software **17**.

Williams, L. A., E. N. Wiebe, et al. (2002). "In Support of Pair Programming in the Introductory Computer Science Course." Computer Science Education.