Building Pair Programming Knowledge through a Family of Experiments

Laurie Williams[1], Charlie McDowell[2], Nachiappan Nagappan[1], Julian Fernald[3], Linda Werner[2]

[1] Department of Computer Science, North Carolina State University, Raleigh, NC 27695 USA
{lawilli3,nnagapp}@unity.ncsu.edu
[2] Department of Computer Science, University of California, Santa Cruz, CA 95064 USA
{charlie, linda}@cs.ucsc.edu
[3] Department of Psychology, University of California, Santa Cruz, CA 95064 USA
jfernald@ucsc.edu

Abstract

Pair programming is a practice in which two programmers work collaboratively at one computer on the same design, algorithm, code, or test. Pair programming is becoming increasingly popular in industry and in university curricula. A family of experiments was run with over 1200 students at two US universities, North Carolina State University and the University of California Santa Cruz, to assess the efficacy of pair programming as an alternative learning technique in introductory programming courses. Students who used the pair programming technique were at least as likely to complete the introductory course with a grade of C or better when compared with students who used the solo programming technique. Paired students earned exam and project scores equal to or better than solo students. Paired students had a positive attitude toward collaboration and were significantly more likely to be registered as computer science-related majors one year later. Our findings also suggest that students in paired classes continue to be successful in subsequent programming classes that require solo programming.

1. Introduction

Pair programming [15] refers to the practice whereby two programmers work together at one computer, collaborating on the same design, algorithm, code, or test. The pair is made up of a *driver,* who actively types at the computer or records a design; and a *navigator*, who watches the work of the driver and attentively identifies problems and makes suggestions. Both are also continuous brainstorming partners.

Computer science faculty are increasingly experimenting, either formally or informally, with pair programming in the classroom. Initial research results [5, 16, 19] indicate that pair programmers produce higher quality code in about half the time of solo programmers. These findings are based on experiments conducted at the University of Utah in a senior-level Software Engineering course. The focus of that research was the affordability of pair programming and the ability of the practice to yield higher quality code without significant increases in time/cost. The researchers also observed educational benefits for student pair programmers. These benefits include higher scores on graded assignments, increased satisfaction and confidence, reduced student frustration, and reduced workload for the teaching staff.

These observations inspired further research examining the use of pair programming in educating computer science students. Educators at the University of California Santa Cruz (UCSC) [4, 6] and North Carolina State University (NCSU) [8, 17, 18] have experimented with pair programming in introductory undergraduate programming courses. This paper compares the experiments at these two US universities involving over 1200 students. To date, we have observed that pair programming has a positive impact on multiple aspects of student performance and enjoyment; none of our findings suggest that student learning is compromised [6-8, 17].

In subsequent sections of this paper, we provide background on this family of experiments and compare the key results from these experiments. Section 2 describes the hypotheses of the studies. Section 3 gives an overview of previous research done on pair programming with students. Section 4 describes the experiments carried out at NCSU and UCSC. In Section 5, we discuss some of the key findings from the two experiments. Section 6 outlines the conclusions and future work.

2. Hypotheses

The NCSU and UCSC experiments were specifically aimed at examining the effects of pair programming in introductory programming courses. At both universities, we tested the six hypotheses listed in Table

1. Each of these hypotheses are examined later in Section 5.

Table 1: Hypotheses

No.	Hypothesis
H1	An equal or higher percentage of students in paired labs will complete the class with a grade of C or better compared to solo programmers.
H2	Students who work in pairs will earn exam scores equal to or higher than solo programming students.
H3	Students who complete programming projects using pair programming produce better programs than students working alone.
H4	Students in paired labs enjoy pair programming and will have a positive attitude towards collaborative programming settings.
H5	The use of pair programming in an introductory computer science course does not hamper student performance in future solo programming courses.
H6	Students participating in pair programming will be significantly more likely than solo programmers to pursue computer science-related majors one year later.

Drawing general conclusions from empirical studies in software engineering is difficult because any process depends to a large degree on a potentially large number of relevant context variables. For this reason, we cannot assume a priori that the results of a study generalize beyond the specific environment in which it was conducted [2]. Researchers become more confident in a theory when similar findings emerge in different contexts [2]. By performing multiple case studies and/or experiments and recording the context variables of each case study, researchers can build up knowledge through a family of experiments [3] which examine the efficacy of a new practice. Replication of experiments addresses threats to experimental validity. We add to the knowledge about pair programming with our results.

As Basili et al. state [2], in order for others to determine whether results from related experiments can be generalized, experimenters need to document the "key choices made during experimental design" and the key indicators of the context in which the experiment took place. The research at NCSU and UCSC involved replication of a family of pair programming experiments in two different university contexts with varying classroom situations. We place information about these two university contexts and classroom situations in Table 2.

Table 2: University Context

NCSU	UCSC
Southeastern US	Western US
Research I University with $440M external funding in 2001-2	Research I University with $68M external funding in 2001-2
77% undergraduates	90% undergraduates
University population: 29,637	University population: over 14,000
CS located in College of Engineering	CS located in College of Engineering
39 tenured/tenure-track faculty and 5 lecturers in CS department	18 tenured/tenure-track faculty and 4 lecturers in CS department
Approximately 900 CS undergraduates (13% women; 24% minorities)	Approximately 550 CS undergraduates (18% women; 48% minorities)
Intro. programming course taught primarily by lecturers with MS in CS	Intro. course taught by lecturers with Ph.D. and tenure/tenure-track faculty
Intro. programming course laboratories taught by more advanced undergraduates	Intro. course laboratories supervised by graduate students and advanced undergraduates

3. Related Work

We have reported previously on the results of the NCSU [8, 17, 18] and UCSC [4, 6, 7] studies in separate papers. Those separate studies each support, to varying degrees, the previously listed hypotheses. In this paper, we focus on the similarities and differences in the findings and experimental conditions in which these studies were conducted.

Researchers at other institutions have also experimented with pair programming. A study performed at the University of Wales [12] indicates that students with lower self-reported programming skill enjoy pair programming more than students with higher self-reported programming skill. Students with a self-reported higher skill level report the least satisfaction when they pair program with students of lesser self-reported skill level. The researchers also found some evidence that students produce their best work when they are paired with a partner of equal skill level. Additionally, an experiment carried out at Carnegie Mellon University [13] in collaboration with the Government of South Africa found that pair programming is as effective as formal inspections in reducing defects in software work products.

Williams et al. found that students working in pairs produce higher quality code in about half the elapsed time when compared with solo programmers [19]. A similar experiment conducted by Nawrocki and

Wojciechowski [9] has concluded contradictory results, indicating that pairs spend almost twice as much total programmer effort as solo programmers. Examining their study, pairs did take twice as much time in the first program which took the students between 2.5-4 hours to complete. These findings are consistent with earlier studies [10, 19] indicating that pairing is significantly more expensive when programmers are first learning the dynamics of pairing. By the end of the fourth program in their study, students had about 13 hours of pairing experience. In the fourth program, the pairs were beginning to look more affordable. Our observations, as well as anecdotal evidence, suggest that it takes programmers a few hours to a few days to transition from solo to collaborative programming [16]. Based on these observations, the Nawrocki study would likely have been more consistent with earlier studies had it been done for a longer period of time.

4. Experimental Methodology and Context

Repetition involves the execution of families of experiments and a set of unifying principles that allows results to be unified and combined [2]. Accordingly, we describe the experiments carried out at NCSU and UCSC. The context varies between these experiments; variables such as class size, class composition, assignments, teaching style, and instructor differ. We examine the consistency of results despite these contextual differences. The consistency attests to the robustness of the results.

We collected data from three semesters of the introductory programming course (hereafter referred to as CS1) at NCSU (Fall 2001, Spring 2002, and Fall 2002), and from three quarters of CS1 at UCSC (Fall 2000, Winter 2001, and Spring 2001). The experimental conditions at each institution are compared in Table 3 and discussed in the remainder of this section.

4.1. Experimental Subjects

Most students are from the College/School of Engineering and are either freshmen or sophomores; however, students of all undergraduate and graduate levels may take the course. The NCSU studies were specifically aimed at the effects of pair programming on beginning students. Therefore, only the results of the first year students and sophomores taking the class for credit were analyzed, discarding data from the lifelong education students, juniors, seniors, and graduate students and auditors from the study. The UCSC studies took into account all the enrolled students in the course.

Table 3: Comparing Experimental Conditions

NCSU	UCSC
15 week semesters	10 week quarters
Two 50 minute lectures per week.	Two 105 minute or three 70 minute lectures per week.
Lecture size during study: 81-206	Lecture size during study: 102-134
One three-hour closed lab per week. Specific closed-lab assignment completed weekly. Mandatory pairing for programming project began during lab.	One 90 minute open lab per week. No specific lab assignment. Students work on programming projects.
Three programming projects (four in Spring 2002) completed outside of lab.	Four or five programming projects completed outside of lab (and worked on during open lab).
Only freshmen and sophomores included in study data[1].	All enrolled[2] students included in study data.
660 students in the study	555 students in the study
Exams were completed individually.	Exams were completed individually.
Grade based on two midterms, a final, lab assignments, and programming projects.	Grade based on four biweekly quizzes, a final, and programming projects.
Pairing and non-pairing sections during same term.	Pairing and non-pairing sections in different terms.
Two instructors taught at least one paired and one solo section each term and one instructor taught only a solo section.	One instructor taught a paired section one term and a solo section in a different term. Two different instructors taught the other two pair sections.
Partners randomly assigned and changed every 2-3 weeks.	Partners assigned from student preference list of three and remained the same all quarter.

4.2. Experimental Lecture Sections

At NCSU, multiple lecture sections are offered each semester. During each semester, at least one lecture section offered traditional, solo programming labs. In the other section(s), students were required to complete their lab assignments using pair programming. At the time of enrolment, students had no knowledge of the experiment or if their section would have paired or solo labs.

[1] Students had the option to participate in the study. Whether or not they opted to participate, they had no choice on whether to pair or work solo.

[2] All enrolled students were invited to participate in the study. A very small percentage elected to not participate which meant they didn't complete the class questionnaire and didn't allow access to their full university record.

At UCSC, only the winter 2001 term offered multiple lecture sections. We thought it was unfair to the students to split one lecture section into pairing and solo labs, therefore, all students in a given term were either pairing or all were solo. The Fall 2000 and Spring 2001 lectures were taught by C. McDowell. The two Winter 2001 lecture sections were taught by tenured/tenure track faculty not otherwise involved in the study.

4.3. Labs and Programming Assignments

The compulsory closed labs at NCSU allowed for the controlled use and observation of pair programming. Closed labs are excellent for controlled use of pair programming [4] because the instructor or teaching assistant can ensure that people are, indeed, working in pairs at one computer. They can also monitor that the roles of driver and navigator are rotated periodically. The lab period is run as a closed lab, where students are given weekly assignments to complete during the allotted time. Lab assignments are "completion" assignments whereby students fill in the body of methods in a skeleton of the program prepared by the instructor. In contrast, the labs at UCSC are open. Instead of specific lab assignments, the students use the lab time primarily as teaching assistant office hours and as a time to get help with their programming projects. At UCSC, there was essentially no direct supervision of the pairing process.

At both universities, the programming projects are "generative," that is, the students start the project from scratch without any structure imposed by the instructor.

4.4. Partner Assignment

At UCSC, students in the pairing sections submitted a list of three names of potential partners, and partners were assigned based on these preferences. In nearly all instances, students were assigned a partner from their list. Those that stated no preference were randomly assigned a partner. Whenever possible, students remained with the same partner throughout the quarter. Due to schedule changes and drops, a few partner reassignments were necessary. As a result of hardships, such as heavy work schedules or living far from campus, 17 students across the three pairing sections were permitted to program alone. Data from these students was combined with the data from the students in the non-pairing section.

In the pair programming labs at NCSU, students were randomly assigned partners based on a web-based computer program; pair assignments were not based on student preferences. Students worked with the same partner for two to three weeks. If a student's partner did not attend a particular lab, then after ten minutes, the student was assigned to another partner. If there were an odd number of students, three students worked together; no one worked alone.

At NCSU, the policy for pairing on projects completed outside of class was changed each semester. Instructors struggled with this policy because they were concerned with student pairing when not under the watchful eye of a lab instructor and fairness between the solo and paired course sections. In Fall 2001, the instructor made pairing on outside projects optional for both sections. The following semester, a policy was instituted whereby students could only pair on projects if they had earned a grade of 70% or better on midterm examinations. However, students complained because they wanted to pair, particularly those who scored close to 70%. This resulted in another policy change in Fall 2002; students in paired sections were then mandated to complete their outside projects with their assigned partner from the closed lab. This policy seemed to work best and was successfully continued in the spring 2003 semester.

At UCSC, we had the same concern about fairness between pairing and solo programmers. The solution adopted at UCSC was to have all students in one term use pairing and all students in a different term work alone. In this way, pairing students were not compared directly (for grading purposes) with solo students. Some indirect comparison is unavoidable to the extent that precise grading standards are applied across different terms. To help minimize student failure to adhere to proper pair programming procedure (alternating driver and navigator roles), along with each assignment, students submitted a log indicating the amount of time they spent on the assignment. Pairing students were asked to differentiate between time spent driving, navigating, and working alone. The form used to record and submit this log (on paper) included the following reminder:

As a reminder, each partner should "drive" roughly 50% of the time the team is working together, and at most 25% of an individual's effort for an assignment can be spent working alone. Any work done by a solitary programmer must be reviewed by the partner. The object is to work together, learning from each other, not to divide the work into two pieces with each partner working on a different piece.

Table 4 summarizes the enrollment and pair/solo sample size for the various classes along with the instructor (denoted by Inst in Table 4) information.

4.5. Questionnaires

In addition to the normal course data, the students at NCSU were asked to complete a lengthy questionnaire at the end of the semester that was used to study their attitude and perception towards pair programming. In addition, all NCSU students took a programming assessment questionnaire at the beginning of the course that was used to assess their initial programming skill level.

At UCSC, the students completed a lengthy questionnaire at the start of the term and a shorter questionnaire at the end of the term. In addition, at the completion of each assignment, students were asked to report on their level of confidence in their solution, how much they enjoyed working on the assignment, and how satisfied they were with the process.

The data from these questionnaires will be the subject of future papers. In addition, some of the data was used to further confirm the equivalence of the populations from the various sections.

5. Quantitative results

In this section, we review the quantitative results of these experiments. Section 5.1 illustrates the academic equivalence of our experimental samples. The remaining sections examine each of the hypotheses from Table 1.

5.1. Academic Equivalence

In experimental design, it is important to assess the equivalence of the control and experimental groups. Based on important, pre-determined factors, the actual equivalence of the groups is used to determine the proper data techniques to use for sound statistical analysis. In this section, we discuss the academic equivalence of the groups under study at both universities. Most US universities require college applicants to take and report the scores of the three-hour standard college entrance exam, the Scholastic Aptitude Test (SAT). The SAT has a Math test/score and a Verbal test/score. Student SAT scores were used at both universities to assess the equivalence of the pairing and solo sections.

A key measure of mathematical ability is the SAT-Math (SAT-M) score. At NCSU, one-way ANOVAs were conducted for each semester to analyze the variance between the SAT-M scores of the paired and solo sections. There were no significant differences between the pairing sections and the solo sections for the Spring 2002 and Fall 2002 semesters. There was a significant difference in SAT-M scores for pairing students (Mean=662) and solo students (Mean=625) for the Fall 2001 semester $F(1,X) = 5.19$, $p<.018$. To accommodate for the SAT-M difference we use the SAT-M as a covariate in our analysis of the Fall 2001 sections.

At University of California-Santa Cruz, one-way ANOVAs were conducted on the four course sections for three variables: (1) SAT-M, (2) SAT-Verbal (SAT-V), and (3) high school GPA (required of most freshmen applicants for admission to UCSC) or transfer GPA (required of all transfer students). There were no significant differences between sections on SAT-M or

Table 4: Sample Sizes (Inst = Instructor)

Institution /Term	Paired Sections		Solo Sections		Total Section Enrollment
	Participants	Total Enrollment	Participants	Total Enrollment	
NCSU F01 Inst A	44	81	0	0	81
NCSU F01 Inst A	0	0	69	96	96
NCSU S02 Inst A -- Section 1	82	105	0	0	105
NCSU S02 Inst A – Section 2	0	0	76	104	104
NCSU S02 Inst B – Section 1	38	48	26	37	85
NCSU S02 Inst B – Section 2	160	206	0	0	206
NCSC F02 Inst B – Section 3	55	104	0	0	104
NCSU F02 Inst C	0	0	110	165	165
UCSC F00 Inst F	171	171	0	0	171
UCSC W01 Inst D	138	138	0	0	138
UCSC W01 Inst E	115	115	0	0	115
UCSC S01 Inst F	0	0	131	131	131

on high school or transfer GPA. There was a significant effect on SAT-V scores *F*(3, 465) = 5.05, *p*<.005. Further follow-up tests revealed that the average SAT-V scores in one of the sections that required pairing was significantly lower than the two other pairing sections (*Mean* = 521 vs. *Mean* = 559 and *Mean* = 573), but not significantly different than the non-pairing section (*Mean* = 547). Combining the three pairing sections indicated that there was no significant difference in SAT-V scores between all pairers and non-pairers (*Mean* = 553 and *Mean* = 547 respectively), only between pairing sections. As a result, the four sections were considered equivalent across all three variables.

5.2. Success rates

Historically, beginning Computer Science classes have a low success rate. We define success rate as the percentage of students who get a C or above in the course. A grade of "C" or higher is an appropriate definition of success because it is the minimum grade required to satisfy further course prerequisites. We evaluated whether pair programming could help improve the success rate of beginning students in an introductory programming course using the following hypotheses.

H_0: The method of programming (solo, paired) and the success rates in the class are statistically independent.

H_1: The method of programming (solo, paired) and the success rates in the class are statistically dependent.

To test the statistical significance of differences in success rates for the paired group versus the solo group, we performed a chi-square test. The chi-square test is designed to test for independence between two categorical variables [1]. A statistically significant result (p<0.05) indicates that pair programming and success in the class are related. These success rates along with the results of the chi-square test are summarized below in Table 5. These results support the alternative hypotheses above and H1 which proposes that an equal or higher percentage of students in the paired labs will complete the class with a grade of C or better compared to solo programmers.

5.3. Performance on Exams

All examinations at NCSU and UCSC were taken individually. There were two midterms and a final exam each semester at NCSU. At UCSC, there were four biweekly quizzes and a final exam. In this paper we discuss only final exam scores to examine the following hypotheses.

Table 5: Success Rate

	# Paired	% Pairers passing	# Solo	% Solo Passing	Stat. Sign.
NCSU[3]	171	70.76	255	60.00	Yes. χ^2= 5.614, p< 0.023
NCSU-Total[4]	379	64.37	281	59.78	No. χ^2= 1.452, p< 0.228
UCSC[5]	404	72.3	148	62.8	Yes. χ^2= 4.57, p< 0.05
(NCSU+ UCSC)	783	68.45	439	61.73	Yes χ^2= 5.670, p< 0.017

H_0: The method of programming (solo, paired) and the student performance in examinations are statistically independent.

H_1: The method of programming (solo, paired) and the student performance in examinations are statistically dependent.

Because of the difference in mean SAT-M scores for the Fall 2001 sections at NCSU, as reported above in Section 5.1., the SAT-M was used as a covariate for the final exam and we used the ANCOVA test (analysis of covariance) in analyzing the Fall 2001 data only. SAT-M was not used as a covariate in the other classes, since the difference in mean score was not statistically significant. The ANOVA was used for these semesters. In every case, there was no statistically significant difference between the final exam scores for the pairing students and the non-pairing students as shown in Table 6. Thus, these results support the null hypothesis above and refute H2 which suggests that students who work in pairs will earn exam scores equal to or higher than solo programming students.

[3] The results of one instructor in Spring 2002 was removed from this analysis because it was felt the comparison was unfair and skewed the results. This instructor had a very large paired section (198 freshman and sophomores) and a small solo section (26 freshman and sophomores), NCSU S02 Inst B in Table 4 above. Often, small sections have a much higher success rate than large sections. Her success rate was similar (57%) for both of these sections.

[4] The students from the prior footnote are added back in.

[5] The pairing status of three students was not available.

Table 6: Final Exam Performance

	Pair			Solo		
	Mean	**Std. Dev.**	**N**	**Mean**	**Std. Dev.**	**N**
NCSU F01	74.1	16.5	44	67.2	18.4	69
NCSU S02a[6]	70.6	28.8	82	73.2	27.4	76
NCSU S02b	71.9	26.7	198	74.9	28.5	26
NCSU-F02	75.1	15.7	55	67.5	35.6	110
UCSC	75.2	18.9	367	74.4	18.5	119

5.4. Project Scores

In addition to success rates and performance in examinations, programming project scores were examined using an ANOVA test. The following hypotheses were examined.

H_0: The method of programming (solo, paired) and the performance on course projects are statistically independent.

H_1: The method of programming (solo, paired) and the performance on course projects are statistically dependent.

At NCSU for the Fall 2001 and Spring 2002 sections the use of pair programming on the programming projects was not an experimental variable. As indicated in Section 4.1, the pairing arrangements at NCSU were different each semester, varying between optional and mandatory. The pairing versus solo was only for the closed lab portion of the class. Students in all sections were allowed to use pair programming. Although we do not have the exact numbers, we estimate that 70% of the students in all sections used pair programming on the projects.

For the Fall 2002 pairing sections at NCSU and all pairing sections at UCSC, all programming projects were completed using pair programming and pair programming was not permitted for students in the solo sections.

As shown in Table 7, there were no statistically significant differences in overall project scores between the pairing sections and the solo sections for any of the semesters at NCSU. At UCSC, the programming projects completed by pair programmers were much better than the projects completed by individual students. For this analysis, we only looked at those students that remained in the class through the final exam. The difference in program scores was statistically significant using ANOVA, F(1,475)=110.13, $p<.001$. At UCSC and in the Fall 2002 NCSU sections the average project score was higher for the pairing sections. It is possible that the NCSU differences are not significant due to the unstructured pairing arrangements.

[6] In the spring 2002 semester, we had a large sample size of 382 students with two instructors. Since we had two instructors each handling a solo and paired class and a large sample size, the results are presented by instructor to maintain the internal validity of the results presented.

Table 7: Programming Project Performance

	Pair			Solo			Stat Sign
	M (%)	**Std. Dev**	**N**	**M (%)**	**Std. Dev.**	**N**	
NCSU - F01	84.8	N/A	44	73.7	N/A	69	No
NCSU - S02a	84.5	24.9	82	81.5	28.5	76	No
NCSU - S02b	78.0	30.9	198	75.9	37.1	26	No
NCSU - F02	83.6	24.4	55	74.2	39.5	110	No
UCSC	86.8	14.7	358	68.6	22.4	118	p< (0. 001)

The UCSC data and previous research results [9] support the alternative hypothesis and H3 that students completing programming projects using pair programming produce better programs than students working alone. In addition, when pair programming on the projects is optional, students that use pair programming during closed labs perform similar to solo students on programming projects.

5.5. Attitude towards pair programming

At NCSU, at the end of the semester, students were given an optional attitude survey [14] in addition to their course and instructor evaluations. We discuss the results of one survey question in this paper: *If you are in a paired section this semester, will you choose a paired section course in the next semester, given there is a paired section?* This survey data was collected from students in the paired section. Table 8 shows that in the spring 2002 semester only 20% of the students expressed a preference for a solo section in the future. In Fall 2002, this number dropped to just over 15%. The student bias in favor of pair programming for both semesters was statistically significant at a very high confidence level, $p< 0.0001$ for both semesters

Table 8: Would You Choose a Paired Section? (Spring 2002 - Fall 2002)

Number of Respondents (Semester)	Yes	I don't care	No
207 (Spring 2002)	124 (59.9 %)	41 (19.8 %)	42 (20.2%)
71 (Fall 2002)	46 (64.7%)	14 (19.7 %)	11 (15.4%)

Similarly, Table 9 displays the results of one question of a survey at UCSC asked at the end of each programming project: *How much did you enjoy working on this programming assignment?* (1=not at all, 7=very much). The difference between the pairing and solo students was statistically significant using a one way ANOVA, F(1,478)=14.9, $p<0.001$.

Table 9: Enjoyment of Project (1 = very dissatisfied, 7 = very satisfied)

Programming Method (all sections)	Mean	Std. dev.
Paired (N=373)	5.15	1.11
Solo (N=106)	4.69	1.02

These results support our hypothesis that students in paired labs enjoy pair programming and have a positive attitude towards collaborative programming settings **(H4)**. Similar results have also been reported by Sanders [11]. A limitation of these findings is that some of these students might not have had solo programming experience, and hence their choice might have been biased towards pair programming. Research done at the University of Wales [12] indicates that students that view themselves as strong programmers are less satisfied with pair programming and more likely to feel they are being "slowed" down by their partner. This would suggest that these self-proclaimed strong programmers could represent a disproportionately large share of the NCSU students who indicated they would not want to work in a pair programming section in the future. We plan to investigate this hypothesis further, as outlined in section 6.

5.6. Performance in subsequent solo programming courses

Some instructors may be concerned that in courses employing pair programming, some students get a "free ride" by placing the entire workload onto their partner and do not learn the course material. At NCSU, this concern was partially addressed by having students submit feedback on their partners via a peer evaluation system called the Peer Evaluation Tool (PET[7]). These evaluations formed a part of the grading structure and the instructor could then judge what, if any, actions were needed. We surmise that students who felt they were penalized by their peer evaluation may have opted for a future solo section, as shown in Table 8.

To determine if students who pair programmed perform satisfactorily in future programming courses requiring solo programming, we examined the following hypotheses.

H_0: The method of programming (solo, paired) and the performance in future solo programming courses are statistically independent.

H_1: The method of programming (solo, paired) and the performance in future solo programming courses are statistically dependent.

We studied students in CS1 and the follow-on class (hereafter referred to as CS2) at both universities. All computer science majors are required to take these courses, and CS1 is a prerequisite for taking CS2. Almost all computer science majors take CS2 the term following CS1. During the time of this study, CS2 was not taught with students programming in pairs. All students had to work alone for every aspect of the class and collaboration of any form was considered cheating.

At NCSU, only students who took both CS1 and CS2 for a grade were included in this analysis. Data from auditors and credit/no-credit students was excluded from the analyses. To perform the analysis, we defined a student to be in a variant group S if his/her CS2 final grade is more than one third of a grade below his/her CS1 grade. For example, if a student got an A in CS1 and B+ in CS2 then he or she would be placed in Group S. This same student would not belong to group S if he or she got an A in CS1 but got an A- in CS2. Table 10 summarizes these results. Of the Fall 2001 CS1 students taking CS2 in Spring 2002, only six students (21.42%) who were paired in CS1 performed worse in CS2 compared to 12 students (46.15%) in the solo class of CS1 in CS2. These results were very close to being statistically significant ($p<0.054$). Similarly of the students from the Spring 2002 CS1 that took CS2 in Fall 2002 semesters, approximately 26% of the students who were paired performed more poorly in CS2 than CS1; almost 30% of the students who worked alone performed more poorly in CS2 than CS1.

[7] http://pairp.csc.ncsu.edu/default.jsp

Table 10: Students in Group S

Semester	Paired (%)	Solo (%)	Statistical Significance
CS1:Fall 2001 – CS2:Spring 2002	21.42 (6/28)	46.15 (12/26)	*No.* χ^2=3.709, p<0.054
CS1:Spring 2002 – CS2:Fall 2002	26.37 (24/91)	29.50 (18/61)	*No.* χ^2=0.179, p<0.672

At UCSC, we looked at the pass rates of students in CS2. We found that students from the pairing sections were just as likely to pass CS2 on their first attempt as the non-paring students. Furthermore, the students from the pairing sections were more likely to attempt CS2. These results are summarized in Table 11. The difference in attempt rates is statistically significant with $\chi^2(1) = 6.17, p < .05$.

Table 11: Pass Rates for CS2 at UCSC

	Attempt Rates	Pass Rates (on 1st attempt) of Attempters
Pair	76.7%	73.6%
Solo	62.2%	72.4%

Based on these results, we believe that pair programming is not detrimental to a student's performance in future solo programming courses. The data supports the alternative hypothesis and H5 which proposed that pair programming in an introductory Computer Science course does not hamper students' performance in future solo programming courses**.**

5.7. Persistence in Computer Science

In addition to being interested in the effects of programming on student performance and subjective experiences with pairing, we were interested in the effect pairing might have on student persistence in Computer Science related majors. Specifically we wondered if using pairing as a learning tool for beginning programmers would influence students' decisions to major in Computer Science related fields. We examined the following hypotheses:

H_0: The method of programming (solo, paired) and the persistence of students in Computer Science related major are statistically independent.

H_1: The method of programming (solo, paired) and the persistence of students in Computer Science related major are statistically dependent.

Pair programming students were significantly more likely to have declared a Computer Science major than solo programming students one year after CS1, regardless of whether they had initially been planning to major in a Computer Science related major. Our findings are shown in Table 12.

Table 12: Percentage of students declaring a Computer Science major 1 year after CS1

	Paired	Solo	Significance
UCSC	56.9%	33.8%	$\chi^2(1) = 12.18$, $p < .001$
NCSU	25.6%	10.5%	$\chi^2(1) = 7.434$, $p < .006$

These results support the alternative hypothesis and H6 that pair programming students are significantly more likely to pursue Computer Science related majors one year later than solo programming students**.**

6. Conclusions and Future Work

Our research coupled with previous empirical studies [9, 11-13] provides strong support for the following conclusions:

- An equal or higher percentage of pair programming students completed the CS1 class with a grade of C or better when compared with solo programmers.
- Student participation in pair programming will lead to at least similar performance on the exams, when compared with solo programming students.
- Students that use pair programming on programming projects will produce better programs than solo programming students. If pair programming is required only for a closed lab there is no discernable impact on programming projects produced outside of the closed lab.
- Students in paired labs have a positive attitude towards collaborative programming settings.
- Students who use pair programming in an introductory Computer Science course are not hampered in future solo programming courses.
- Students who use pair programming in an introductory programming course are significantly more likely than solo-programming students to pursue Computer Science related majors one year later.

We plan to continue our experimental work testing the impact of pair programming on student performance. Additionally, we plan to investigate the importance of pair compatibility through personality tests like the Myers-Briggs personality test and matching the skill level of students working in pairs. We have reported some gender-specific results [7], but further study is necessary. In addition we plan to gather data for minority students to obtain meaningful results for this important group.

7. Acknowledgements

We would like to thank the members of the Software Engineering reading group at NCSU for their valuable comments while reviewing this paper. The National Science Foundation Grants DUE CCLI 0088178 and EIA-0089989 provided funding for the research in these pair programming experiments. We thank Dr. Heather Bullock, Department of Psychology UCSC for her help on the project at UCSC. We acknowledge Dr. Wiebe, assistant professor in the Mathematics Education department at NCSU, and his students Miriam Ferzli and Julie Petlick. We also thank Ms. Carol Miller, Ms. Suzanne Balik, Mr.Jason Schwarz, and Dr.Matt Stallmann the instructors of the CS1 and CS2 courses for their assistance and guidance and graduate student Kai Yang who began the NCSU study. Finally thanks to Professors Scott Brandt and Alex Pang at UCSC for allowing us to run our experiment on their classes.

8. References

[1] A. Agresti and B. Finlay, *Statistical Methods for the Social Sciences*: Dellen Publishing Company, Collier Macmillan Publishers, Macmillan, Inc., 1986.

[2] V. Basili, Shull, F.,Lanubile, F., "Building Knowledge through Families of Experiments," *IEEE Transactions on Software Engineering*, vol. Vol. 25, No.4, 1999.

[3] V. R. Basili, F. Shull, and F. Lanubile, "Building Knowledge Through Families of Experiments," *IEEE Transactions on Software Engineering*, vol. 25, pp. 456 - 473, 1999.

[4] J. Bevan, L. Werner, and C. McDowell, "Guidelines for the User of Pair Programming in a Freshman Programming Class," presented at Conference on Software Engineering Education and Training, Kentucky, 2002.

[5] A. Cockburn and L. Williams, "The Costs and Benefits of Pair Programming," presented at Extreme Programming and Flexible Processes in Software Engineering (XP2000), Cagliari, Sardinia, Italy, 2000.

[6] C. McDowell, L. Werner, H. Bullock, and J. Fernald, "The Effect of Pair Programming on Performance in an Introductory Programming Course," presented at ACM Special Interest Group of Computer Science Educators, Kentucky, 2002.

[7] C. McDowell, L. Werner, H. Bullock, and J. Fernald, "The Impact of Pair Programming on Student Performance of Computer Science Related Majors," presented at submitted to the International Conference on Software Engineering 2003, Portland, Oregon, 2003.

[8] N. Nagappan, L. Williams, M. Ferzli, K. Yang, E. Wiebe, C. Miller, and S. Balik, "Improving the CS1 Experience with Pair Programming," presented at SIGCSE 2003, 2003.

[9] J. Nawrocki and A. Wojciechowski, "Experimental Evaluation of Pair Programming," presented at European Software Control and Metrics (ESCOM 2001), London, England, 2001.

[10] J. T. Nosek, "The Case for Collaborative Programming," in *Communications of the ACM*, vol. March 1998, March 1998, pp. 105-108.

[11] Sanders. D., "Student perceptions of the suitability of Extreme and Pair Programming," presented at XP Agile Universe, 2001.

[12] L. Thomas, M. Ratcliffe, and A. Robertson, "Code Warriors and Code-a-Phobes: A study in attitude and pair programming," presented at SIGCSE, Reno, NV, 2003.

[13] J. E. Tomayko, "A Comparison of Pair Programming to Inspections for Software Defect Reduction," *Computer Science Education*, vol. September, 2002.

[14] E. Weibe, L. A. Williams, K. Yang, and C. Miller, "Computer Science Attitude Survey," North Carolina State University, Raleigh, NC CSC TR-2003-01, 2003.

[15] L. Williams and R. Kessler, *Pair Programming Illuminated*. Reading, Massachusetts: Addison Wesley, 2003.

[16] L. Williams, R. Kessler, W. Cunningham, and R. Jeffries, "Strengthening the Case for Pair-Programming," in *IEEE Software*, vol. 17, 2000, pp. 19-25.

[17] L. Williams, E. Wiebe, K. Yang, M. Ferzli, and C. Miller, "In Support of Pair Programming in the Introductory Computer Science Course," *Computer Science Education*, vol. September, 2002.

[18] L. Williams, K. Yang, E. Wiebe, M. Ferzli, and C. Miller, "Pair Programming in an Introductory Computer Science Course: Initial Results and Recommendations," presented at OOPSLA Educator's Symposium, Seattle, WA, 2002.

[19] L. A. Williams, "The Collaborative Software Process PhD Dissertation," in *Department of Computer Science*. Salt Lake City, UT: University of Utah, 2000.