

Pair Learning: With an Eye Toward Future Success

Nachiappan Nagappan¹, Laurie Williams¹, Eric Wiebe², Carol Miller¹, Suzanne Balik¹, Miriam Ferzli², Julie Petlick²

¹ Department of Computer Science, North Carolina State University,
Raleigh, NC 27695

{nnagapp, lawilli3, miller, spbalik} @unity.ncsu.edu

² Department of Math, Science and Technology Education, North Carolina
State University, Raleigh, NC 27695

{wiebe, mgferzli, jdhinson} @unity.ncsu.edu

Abstract. Pair programming is a practice in which two programmers work collaboratively at one computer on the same design, algorithm, or code. Prior research indicates that pair programmers produce higher quality code in essentially half the time taken by solo programmers. Pair programming is becoming increasingly popular in industry and in university curricula. An experiment was run at North Carolina State University over a period of one and a half years to assess the efficacy of pair programming as an alternative educational technique in an introductory programming course. We found that the retention rate of the students in the introductory programming courses is equal to or better than that of the students in the solo programming courses. Most students show a positive attitude towards collaborative programming, and students in paired classes continue to be successful in subsequent programming classes that require solo programming. Pair programming also leads to a reduced workload for the course staff in terms of grading, questions answered and teaching effort.

1. Introduction

In industry, software developers generally spend 30% of their time working alone, 50% of their time working with one other person, and 20% of their time working with two or more people [6]. However, most often in an academic environment, programmers must learn to program alone, and collaboration is considered cheating. This time spent working alone unfortunately conflicts with a student's future professional life in which collaboration is both encouraged and required. In addition, studies show that cooperative and collaborative pedagogies are beneficial for students [11, 12].

Research results [5, 17, 20] indicate that pair programmers produce higher quality code in about half the time when compared with solo programmers. These research results are based on experiments held at the University of Utah in a senior-level

Software Engineering course. The focus of that research was the affordability of the practice of pair programming and the ability of the practice to yield higher quality code without significant increases in time/cost. However, the researchers observed educational benefits for the student pair programmers. These benefits included superior results on graded assignments, increased satisfaction, reduced frustration from the students, increased confidence from the students on their project results, and reduced workload for the teaching staff.

These observations inspired further research directed at the use of pair programming in educating Computer Science students. Educators at the University of California-Santa Cruz (UCSC) [4, 8] and North Carolina State University (NCSU) [10, 18, 19] have reported on the use of collaborative pair programming in introductory undergraduate programming courses. The experiments were specifically designed to assess the efficacy of pair programming in an introductory Computer Science course. These researchers have found that pair programming improved the retention rates of the students. We have continued these studies and report our findings in this paper.

In our experiment, specifically aimed at the effects of pair programming on beginning students, we have examined the following four hypotheses related to the introductory course:

H1 An equal or higher percentage of students in paired labs will complete the class with a grade of C or better when compared with solo programmers.

H2 Students in paired labs will have a positive attitude towards collaborative programming settings.

H3 Doing pair programming in an introductory Computer Science course does not hamper students' performance in future solo programming courses.

H4 Student participation in pair programming will lead to a reduced workload for the course staff in terms of grading, questions answered, and teaching effort when compared with the course staff of solo sections.

In subsequent sections of this paper we provide background on this experiment and give detailed analysis of the results. Section 2 gives an overview of the previous research done in pair programming. Section 3 describes the experiment carried out at North Carolina State University. Section 4 and Section 5 present the quantitative and qualitative results. Section 6 outlines the conclusions and future work.

2. Pair Programming

Pair programming [16] refers to the practice whereby two programmers work together at one computer, collaborating on the same algorithm, code, or test. The pair is made up of the *driver*, who actively types at the computer or records a design; and the *navigator*, who watches the work of the driver and attentively identifies problems and makes suggestions. Both are also continuous brainstorming partners.

Pair programming has been practiced for a long time [16], yet only recently has much research been done on its effectiveness. One of the major factors responsible for the growth in popularity of the practice is an emerging software development methodology called Extreme Programming (XP) [3]. XP was designed for the development of small- to medium-size projects. This highly collaborative methodology reduces initial planning stages for projects and places more emphasis on customer-centric activities, adapting to their ever-changing requirements. Planning is done incrementally throughout the development cycle. XP utilizes the pair programming practice for all production code.

Research at UCSC has reported positive results in studies involving pair programming with students [4, 8]. Their studies indicate that pair programming helped increase the retention rate of students who might have otherwise dropped out of the course. They also indicate that pair programming students produce better quality code and perform comparably on exams when compared to solo programming students [9]. In addition, research done at UCSC on pairing protocol issues provide several recommendations, such as pairing students with other students within the same section; pairing students with other students with similar skill level, and having a coding standard [4]. Finally, research done at UCSC on a large sample of students (555 students) indicates that pairing bolsters the course completion and pass rates and leads to higher persistence of students in a computer related major [9]. Furthermore, students show a positive attitude towards pair programming [9].

Research done at the University of Wales [13] indicates that students with lower self-confidence enjoy pair programming the most. Students with a higher skill level report the least satisfaction when they pair program with students of lesser skill level. The researchers also found some initial evidence that students produce their best work when they are paired with a partner of equal skill and confidence level.

The research studies discussed so far involve collocated pair programmers. Additional research was done at NCSU concerning distributed pair programming, whereby the programmers are not physically collocated. An experiment was run in a graduate-level Computer Science course, Object Oriented (OO) Languages and Systems. The focus of this experiment was the performance of collocated and distributed, pair and solo programmers. The initial results indicate that distributed pair programming is a feasible technique for OO development [2]. Though there were no statistically significant results, there were three initial indications from the study. The quality of code written by the distributed pairs was comparable to collocated pairs and distributed non-pairs. There were similar productivity results (measured by the lines of code) among the collocated and distributed pairs [2]. Finally, the results also indicated that distributed pair programming fosters teamwork and communication within a virtual team.

3. Experiment

A structured empirical study was run at NCSU for three semesters (Fall 2001, Spring 2002 and Fall 2002). For these research studies, we chose an introductory programming course, Introduction to Computing – Java (CSC116). The following factors led to the choice of this course:

The course is an introductory programming course that is required of all Computer Science (CSC) majors. CSC 116 is also offered as a service course for students from other departments and for lifelong education students.

The course has compulsory closed labs, which allow for the controlled use and observation of pair programming. Closed labs are excellent for controlled use of pair programming [4]. The instructor or teaching assistant can ensure that people are, indeed, working in pairs at one computer. He or she can also monitor that the roles of driver and navigator are rotated periodically.

The course is taught with two 50-minute lectures and one three-hour lab each week. Students attend labs in groups of 24 with others in their own lecture section. 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. Student grades are based on two midterm exams, one final exam, lab assignments, and programming projects that are completed outside of the closed lab. The programming projects are generative, that is, the students start the project from scratch without any structure imposed by the instructor. Most students are from the College of Engineering and are either freshmen or sophomores. However, students of all undergraduate and graduate levels may take the course.

Our study is specifically aimed at the effects of pair programming on beginning students. Therefore, we analysed the results of the freshmen and sophomores only. We eliminated the lifelong education students, juniors, seniors, and graduate students from the study. We also only analysed the data from students who took the course for a grade, concluding that students who audited the class or took it for credit only were not as motivated to excel as other students.

The Fall 2001 experiment was run in two sections of the course; the same instructor taught both sections. Additionally, the midterm exams and the final exam were identical. One section had traditional, solo programming labs. In the other section, students were required to complete their lab assignments by participating in the pair programming practice. When students enrolled for the class, they had no knowledge of the experiment or if their section would have paired or solo labs. In the pair programming labs, 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 show up for a particular lab, 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.

In the Spring 2002 the same experiment was carried out on a much larger scale due to the increased sample size available during the semester. Two instructors handled the CSC 116 (CS1) course. The fall 2002 semester also employed the same model of operation of classes as in the previous semesters, with two instructors teaching the course. Also, in the spring semester of 2002, we analyzed the performance of students in the follow-on class (CS2) who had pair programmed in CS1 the previous fall semester. Similarly, we analyzed the performance of Fall 2002 CS2 students who has taken CS1 in Spring 202

The course also included programming projects that require work outside of the closed lab. In Fall 2001, we gave the students in both sections the option of working alone or in pairs for these projects. We modified this in Spring 2002 where only students who attained a score of 70% or better on the exams could opt to pair. At that time, we felt those who did not attain a score of 70% or above should not work with a pair on the project lest they rely too heavily on their partner to produce the project. Most students who were eligible to pair chose to pair program on projects. However, the instructors now feel that the 70% eligibility might be unfair to the students, and this practice was discontinued starting in Fall 2002. At this time, we began to assign mandatory partners; students worked on projects with the same partner assigned in labs.

4. Quantitative results

We analyzed the quantitative data semester by semester because different instructors with different course presentation styles and different exam/project content handled the course across the three semesters. In the spring 2002 semester, we had two instructors each handling a solo and paired class and a large sample size. Therefore, the results are presented by instructor for Spring 2002.

4.1 Academic Equivalence

4.1.1 Prior Programming experience

We wanted to determine if we had academic equivalence in our experimental groups. We assessed academic equivalence in two ways: a programming assessment (which is reflective of their prior programming experience) and the students' SAT-M scores (which are reflective of the students' mathematical ability). All students took a programming assessment questionnaire at the beginning of the semester. The questionnaire contained some very basic questions on programming. The main purpose of the assessment was to examine the differences in the programming background the students had before they took the course. This analysis helped us determine whether any section had more knowledge about programming compared to other sections

before the start of the course. This assessment did not carry any weight in the grading of the course. There were eight questions in the questionnaire that covered the areas of computer arithmetic, operator precedence, selection, iteration, nested loops, arrays, encapsulation, and functions. The programming assessment was evaluated out of a total score of eight points. We tested the statistical significance of the difference in programming assessment scores by using a t-test. Table 1 shows our observations across the Fall 2001, Spring 2002, and fall 2002 semesters.

The hypotheses for the t-test are as follows:

H_0 : There is no significant difference between the two sections in terms of prior programming knowledge.

H_1 : There is a significant difference between the two sections in terms of prior programming knowledge.

If we get statistically significant results ($p < .05$), we accept the alternative hypotheses (H_1) else we accept the null hypotheses (H_0). Hence from the t-test results in Table 1, the two sections were equivalent in terms of their programming assessment scores for the fall 2001 semester only. Hence we further examine the students SAT-M scores to study their academic equivalence.

Table 1: Programming Assessment Score

Semester	Paired Mean	Paired Std Dev	Solo Mean	Solo Std Dev	Stat. Significance
Fall 2001	2.69	2.22	1.77	1.62	No, t=1.807, p<0.080
Spring 2002	2.05	2.05	1.62	1.61	Yes, t=3.51, p<0.0006
Fall 2002	2.55	1.81	1.64	1.61	Yes, t=2.08, p<0.0389

4.1.2 Math Skills

We use the students' SAT-M scores to form the basis of our math skill analysis. We use a one-way ANOVA test to analyze the variance between the SAT-M scores of the paired and solo sections to investigate the academic equivalence of the two groups with respect to their math skills. As shown in Table 2, the ANOVA between SAT-M

scores yielded statistically significant results ($p < .05$) in the fall semester 2001. The mean SAT-M scores are essentially equivalent across the other two semesters.

Table 2: SAT-M scores and statistical significance results

Semester	Paired Mean	Solo Mean	Statistically Significant
Fall 2001	662.10	625.43	Yes ($F=5.19, p < 0.018$)
Spring 2002	634.88	640.25	No ($F=1.101, p < 0.416$)
Fall 2002	639.12	640.15	No ($F=1.088, p < 0.395$)

The statistical results in the spring 2002 and fall 2002 semesters were rejected at a very high level of confidence. We consider the level of academic equivalence in our analysis.

4.2 Success rates

Historically, beginning Computer Science classes have a low success rate, often cited informally as about 50% nationally. Success rate is defined as students who get a C or above in the course. We chose C because it is the minimum grade required in a course to satisfy further course prerequisites. We evaluated whether pair programming could help improve the success rate of beginning students in an introductory programming course. To test the statistical significance of the difference in success rates, we performed a chi-square test. The chi-square test is designed to test for independence between two categorical variables [1].

The hypotheses for the chi-square test are as follows:

H₀: There is a statistical independence between the method of programming (solo and paired) and the success rates in the class.

H₁: There exists a statistical dependence between the method of programming (solo and paired) and the success rates in the class.

A statistically significant result ($p < 0.05$) indicates that pair programming impacted the student final grades. These success rates along with the results of the chi-square test are summarized below in Table 3.

Table 3: Success Rate

Semester-Section	C and above (# of students)	Below C (# of students)	Success Rate	Statistically Significance
Fall 2001 ¹ -Paired	30	14	68.18%	Yes $\chi^2 = 5.849$, $p < 0.016$
Fall 2001-Solo	31	38	44.93%	
Spring 2002-Paired (Inst 1)	54	28	65.85 %	No $\chi^2 = 0.000$, $p < 0.993$
Spring 2002-Solo (Inst 1)	50	26	65.79 %	
Spring 2002-Paired (Inst 2)	113	85	57.07 %	No $\chi^2 = 0.004$, $p < 0.952$
Spring 2002-Solo (Inst 2)	15	11	57.69 %	
Fall 2002-Paired	47	8	85.45 %	Yes $\chi^2 = 7.292$, $p < 0.007$
Fall 2002-Solo	72	38	65.45 %	

Hence, from the results obtained over the past three semesters we see the final course grades obtained by students working in pairs is equal or better than those working solo in two of the three semesters. Thus, two out of three semesters validate our first hypotheses 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 (H1)*.

Instructor 2 from Spring 2002 handled the Fall 2002 paired class. She attributed the increase in success rate from 57.07% in Spring 2002 to 85.45% in Fall 2002 to the following factors: a new textbook was introduced which was easier to understand from the students viewpoint; daily in-class exercises were done along with a partner; required attendance in class worth 5% of the overall grade, and written homework exercise from the book that urged the students to read the textbook.

¹ The Fall 2001 semester results have previously been reported in [10, 18, 19, 21] .

4.3 Attitude towards pair programming

At the end of the semester, along with their course and instructor evaluations, students were given an optional attitude survey [15]. Using the survey, we tried to determine the students' attitudes towards pair programming. The entire survey had 63 questions. We discuss the results of only one of these questions 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 in the spring 2002 and fall 2002 semesters from students in the paired section. Students in the solo section would not have had an informed opinion on this question. Table 4 shows that in the spring 2002 semester the majority of the students (approximately 80%) did not express a preference for a solo section in future. Over 84% of the Fall 2002 students did not express a preference for a solo section in future.

Table 4: Attitude survey Results (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%)

Results from Table 4 support our initial hypotheses that *students in paired labs have a positive attitude towards collaborative programming settings (H2)*. 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. Similar to research results [13], we surmise that the 20% of students in the spring 2002 semester and the 15% of students in the fall 2002 semester who did not want to work in a future pair programming section might have had a higher skill level and would not have liked being "slowed" down by their partner. We will investigate this hypothesis further, as outlined in section 6.

4.4 Performance of paired students in solo programming courses in future semesters

Some instructors may be concerned that in courses employing pair programming, several students get a "free ride" by passing on the entire workload to their partner and do not learn the course material. We partially addressed this problem by having students submit feedback on their partners via an effective peer evaluation system called the Peer Evaluation Tool (PET²). These evaluations formed a part of the

² <http://pairprogramming.csc.ncsu.edu/pairlearning/testing/mystudentlogin.jsp>

grading structure, and the instructor could then judge what action needed to be taken upon identification of the problems.

To show that students who pair programmed perform satisfactorily in future programming courses and that pair programming was not detrimental to students who program in solo courses in the future, we examined students in CS1 and the follow-on class CS2. All CSC majors are required to take these courses, and CS1 is a prerequisite for taking CS2. Almost all CSC majors take CS2 the following semester after taking CS1. CS2 is “Programming Concepts – Java”. During the time of this reported study, CS2 was not taught with students programming in pairs. All students had to work alone, and collaboration of any form was considered cheating.

4.4.1 Fall 2001-Spring 2002 semesters

First, we analyzed students who took CS1 in Fall 2001 and CS2 in Spring 2002. We obtained the results shown in Table 5. In this table, “Paired” refers to the students in the paired section and “Solo” refers to students who worked solo in CS1 in the fall semester. In CS2, all students worked alone.

Table 5: Performance in CS2 (Spring '02) of students in CS1 (Fall '01)

Section	Percentage of A's, B's in CS2	Percentage of C's, D's, F in CS2
Paired in CS1 (N=33)	69.70% (23/33)	30.30% (10/33)
Solo in CS1 (N=29)	44.82% (13/29)	55.18% (16/29)

From Table 5, more students who paired in CS1 earned a grade of A or B. In order to statistically quantify these results, we run a chi-square test with the following hypotheses.

H₀ : Performance in future programming courses are independent of programming technique employed the previous semester.

H₁ : Performance in future programming courses are dependent of programming technique employed the previous semester.

The results were statistically significant, ($\chi^2=3.921$ ($p<0.048$)) indicating that the performance in future programming courses is impacted by the technique employed in the previous semester.

Analyzing further, a student is said to be in a variant Group S if his/her CS2 final grades are more than one third of a grade below their CS1 grades. 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 6 summarizes these results. 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.

Table 6: Students in Group S

Section	Group S	Variant group S's performance rate
Paired (N=28) ³	6	21.42 %
Solo (N=26) ⁴	12	46.15 %

4.4.2 Spring 2002-Fall 2002 Semester

In the fall 2002 semester we obtained the following results as shown in Table 7. Students who had programmed solo in the previous semester performed much better than students who pair programmed. A chi-square test resulted in statistically significant results ($\chi^2=3.934$, $p<0.047$), thereby revealing that there was a dependence relationship between the student performance and the technique of programming employed the previous semester. However, it must be noted that the percentage of students earning an A or B was higher than for either of the groups the prior semester (see Table 5).

Table 7: Performance in CS2 (Fall '02) of students in CS1 (Spring '02)

Section	A's, B's in CS2 (# of students)	C's, D's, F in CS2 (# of students)
Paired in CS1 (N=91)	71.43% (65/91)	28.57% (26/91)
Solo in CS1 (N=66)	85.25% (52/61)	14.75% (9/61)

Table 8 shows that approximately 26% of the students who were paired did worse in CS2 than CS1; almost 30% of the students who worked alone did worse in CS2 than CS1. Comparatively in the previous semester only 21.42% of the paired students did worse in CS2 relative to CS1.

³ Five students were dropped from the analysis, thereby reducing the sample size to 29. Two of them decided to drop the course and three of them took CS2 for credit only.

⁴ Three students were dropped from the analysis thereby reducing the sample size to 26 as they took CS2 for credit only.

Table 8: Students in Group S

Section	Group S	Variant group S's performance rate
Paired (N=91)	24	26.37 %
Solo (N=61)	18	29.50 %

Based on the results listed above, we can say that pair programming is not detrimental to a student's performance in future programming courses done in solo. Hence we conclude that *pair programming in an introductory Computer Science course does not hamper students' performance in future solo programming courses. (H3).*

5 Qualitative results

Qualitative results in CSC 116 [7] were obtained by observing the CSC 116 laboratory sessions. These observations were followed up with further focus groups with the students (drawn from the entire student sample) and the Lab Instructors (LIs). Additionally, in the fall 2002 semester, we quantified some of our in-class observations. These results are presented in the following section.

5.1 Students

Students expressed a preference for being able to ask their partners questions immediately as problems arise rather than having to wait for an LI to answer them. Having someone there while working on problems seemed to help students clarify ideas, pick up on minor errors, and work on understanding conceptual knowledge.

In solo labs, most of the students had to wait for 10-30 minutes to get their questions answered, and sometimes they would not be able to get their questions answered because the LIs would be busy answering other students' questions. These students would give up and continue working, ignoring their mistakes for the time being. Comparatively, since the pairs were self-sufficient, lab instructors had more time to get around to needy students than in the unpaired sections. Paired students who needed help found it easy to get help from the LI and had little down time [18].

In the focus groups, students described "partner compatibility" as the number one problem to address in paired labs. We intend to address this compatibility issue in the following semesters. We have already carried out some compatibility research in the NCSU undergraduate Software Engineering course. We will use these results to better understand "partner compatibility." Despite their frustrations with compatibility issues, students expressed their understanding that pair programming would help them in future professional work environments where people are often randomly matched to collaborate on programming projects.

Additionally, observations were made in nine paired lab sections and seven solo lab sections [14]. Students raised their hands when they had a question. When many students need questions answered, those students waiting would often give up, leaving them with unresolved problems. These are referred to as ‘Give Ups’ in Table 9. As shown in Table 9, the paired lab averaged a little over one ‘Give Up’ every lab, while the solo labs averaged almost two and an half times more than the paired lab. This demonstrates that students are better served by the LIs in paired labs.

Table 9: ‘Give Ups’ statistics for Fall 2002 [14]

Section	No. of ‘Give ups’ Mean	No. of ‘Give Ups’ Standard Deviation
Paired	1.14	0.90
Solo	3.17	2.56

5.2 Lab Instructors

One benefit of pair programming is that grading is reduced by a factor of two for projects and labs. Additionally, in solo lab sections, the LIs were often overwhelmed with questions. LIs often spent a minimum of five minutes and a maximum of 20 minutes with each student. Most of the students had basic questions regarding the syntax, function passing and compilation errors. [7]

In the focus groups, the LIs also felt that the quality of questions asked by the students in the paired class were indicative of higher-order learning when compared with those of students in the solo programming class. Students in the solo programming class were always asking basic questions, for example, regarding the syntax. Paired students, spent more time discussing advanced issues with their LIs [18] For example, students in paired labs would ask the LIs how to improve their algorithm or how to apply it to another scenario. Finally LIs observed that paired students’ efforts and willingness to learn seemed to surpass their “traditional” counterparts. The solo lab students needed to be taught every step of the way.

To illustrate, we report results of counting the average number of questions asked in nine paired lab sections and seven solo lab sections [14]. Students in the solo programming class asked more questions and the standard deviation also is very large, which indicates that the number of questions was widely variable between labs (see Table 10). By looking at the mean number of questions we can say that the LIs had less questions to answer in the paired labs, which ensures that the response time of the LIs to student questions was better in the paired labs.

Table 10: Statistics of questions asked (Fall 2002) [14]

Section	Total Questions Mean	Total Questions Standard Deviation
Paired	43.11	7.47
Solo	56.14	30.68

Thus from the above evidence results we can conclude that student participation in pair programming will lead to a reduced workload in terms of grading, questions answered, and teaching effort for the course staff when compared with the teaching staff for students who worked solo (**H4**). This thus validates our initial hypotheses.

6. Conclusions and Future Work

Our study provides strong results of the following findings:

- 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.
- Students in paired labs have a positive attitude towards collaborative programming settings.
- Students pair programming in an introductory Computer Science course does not hamper students' performance in future solo programming courses..
- Student participation in pair programming will lead to a reduced workload in terms of grading, questions answered, and teaching effort for the course staff when compared with the teaching staff for students who worked solo.

We will continue the experiment in the year 2003 and use the data to further validate our claims. Also we will further investigate pair compatibility by using personality tests like the Myers-Briggs personality test and using the skill level of students to match pairs. Already, a trial experiment has been run in the spring semester 2002 in the undergraduate Software Engineering class at NCSU to match students according to personality profiles and skill level. We will obtain similar results in the CSC 116 course. We will gather results for minority and female students to obtain meaningful results for these important groups.

Acknowledgements:

We would like to thank the members of the Software Engineering reading group at NCSU for their valuable comments while reviewing this paper. We would also like to thank Dr. Matt Stallmann of the Computer Science Department, NCSU for providing us

with access to the grades of the CS2 course in Spring 2002. The National Science Foundation Grant DUE CCLI 0088178 provided funding for the research in this pair programming experiment.

References:

- [1] Agresti, A. and Finlay, B., *Statistical Methods for the Social Sciences*: Dellen Publishing Company, Collier Macmillan Publishers, Macmillan, Inc., 1986.
- [2] Baheti, P., Williams, L., Gehringer, E., and Stotts, D., "Exploring Pair Programming in Distributed Object-Oriented Team Projects," Proceedings OOPSLA Educator's Symposium, Seattle, WA, 2002.
- [3] Beck, K., *Extreme Programming Explained: Embrace Change*. Reading, Massachusetts: Addison-Wesley, 2000.
- [4] Bevan, J., Werner, L., and McDowell, C., "Guidelines for the User of Pair Programming in a Freshman Programming Class," Proceedings Conference on Software Engineering Education and Training, Kentucky, 2002.
- [5] Cockburn, A. and Williams, L., "The Costs and Benefits of Pair Programming," Proceedings eXtreme Programming and Flexible Processes in Software Engineering - XP2000, Cagliari, Sardinia, Italy, 2000.
- [6] DeMarco, T. and Lister, T., *Peopleware*. New York: Dorset House Publishers, 1977.
- [7] Ferzli, M., Wiebe, E., and Williams, L., "Paired Programming Project: Focus Groups with Teaching Assistants and Students," North Carolina State University, Raleigh, NC CSC TR-2002-16, 2002.
- [8] McDowell, C., Werner, L., Bullock, H., and Fernald, J., "The Effect of Pair Programming on Performance in an Introductory Programming Course," Proceedings ACM Special Interest Group of Computer Science Educators, Kentucky, 2002.
- [9] McDowell, C., Werner, L., Bullock, H., and Fernald, J., "The Impact of Pair Programming on Student Performance of Computer Science Related Majors," Proceedings submitted to the International Conference on Software Engineering 2003, Portland, Oregon, 2003.
- [10] Nagappan, N., Williams, L., Miriam Ferzli, Yang, K., Wiebe, E., Miller, C., and Balik, S., "Improving the CS1 Experience with Pair Programming," Proceedings SIGCSE 2003, 2003.
- [11] Slavin, R., *Using Student Team Learning*. Boston: The Center for Social Organization of Schools, The Johns Hopkins University, 1980.
- [12] Slavin, R., *Cooperative Learning: Theory, Research and Practice*. New Jersey: Prentice Hall, 1990.
- [13] Thomas, L., Ratcliffe, M., and Robertson, A., "Code Warriors and Code-a-Phobes: A study in attitude and pair programming," Proceedings SIGCSE, Reno, NV, 2003.
- [14] Wiebe, E., Williams, L. A., Petlick, J., Nagappan, N., Balik, S., Miller, C., and Ferzli, M., "Pair Programming in Introductory Programming Labs," Proceedings Submitted

to American Society for Engineering Education Annual Conference and Exposition 2003, 2003.

- [15] Weibe, E., Williams, L. A., Yang, K., and Miller, C., "Computer Science Attitude Survey," North Carolina State University, Raleigh, NC CSC TR-2003-01, 2003.
- [16] Williams, L. and Kessler, R., *Pair Programming Illuminated*. Reading, Massachusetts: Addison Wesley, 2003.
- [17] Williams, L., Kessler, R., Cunningham, W., and Jeffries, R., "Strengthening the Case for Pair-Programming," in *IEEE Software*, vol. 17, 2000, pp. 19-25.
- [18] Williams, L., Wiebe, E., Yang, K., Ferzli, M., and Miller, C., "In Support of Pair Programming in the Introductory Computer Science Course," *Computer Science Education*, vol. September, 2002.
- [19] Williams, L., Yang, K., Wiebe, E., Ferzli, M., and Miller, C., "Pair Programming in an Introductory Computer Science Course: Initial Results and Recommendations," Proceedings OOPSLA Educator's Symposium, Seattle, WA, 2002.
- [20] Williams, L. A., "The Collaborative Software Process PhD Dissertation," in *Department of Computer Science*. Salt Lake City, UT: University of Utah, 2000.
- [21] Yang, K., "Masters Thesis: Pair learning in undergraduate computer science education," in *Computer Science*. Raleigh, NC: North Carolina State University, 2002.