

## Pair Programming in Introductory Programming Labs

Eric N. Wiebe, Laurie Williams, Julie Petlick, Nachiappan Nagappan,  
Suzanne Balik, Carol Miller and Miriam Ferzli  
NC State University, Raleigh, NC

*ABSTRACT: This project looks at the practice of pair programming as a vehicle for improving the learning environment in introductory computer science labs, a nearly universal course for all engineering students. 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. A multiyear project is currently underway at North Carolina State University, looking at the efficacy of pair programming in an introductory Computer Science (CS1) course. Results indicate that student pair programmers are more self-sufficient in lab and are more likely to complete the class with a grade of C or better. The effect of pair programming on specific graded exams and projects are less clear. Demographic data, prior achievement scores, and current course scores were brought together with interview data, attitude surveys, and in-lab observations to guide the evaluation. As part of the in-lab data collection, novel techniques are being developed to better understand pair dynamics and student/instructor interactions. Collectively, these evaluative methods have guided the iterative implementation of paired programming instructional methods. Current challenges being addressed include lab instructor training, student/instructor concerns over equity in effort on assignments, pair dynamics in lab, and collaborative logistics of pair programming outside of lab.*

### I. Introduction

Extreme Programming (XP) <sup>1</sup> has popularized a structured form of programmer collaboration called pair programming. Pair programming is a style of programming in which *two* programmers work side-by-side at *one* computer, continuously collaborating on the same design, algorithm, code, or test. One of the pair, called the *driver*, is types at the computer or writes down a design. The other partner, called the *navigator*, has the task of monitoring the driver's work for tactical and strategic defects. Tactical defects are local errors such as syntax errors, typos, or improper method calls. Strategic errors involve, as the name implies, problems in strategy, leading incorrect or inefficient solutions to programming problems. Because the navigator is not involved in the highly focused work of typing the code, they can afford to look ahead and monitor overall strategy. Still, the driver and navigator continuously spend time brainstorming and negotiating their course of action.

Ideally, communication between the driver and navigator is regular with the driver explaining what he or she is doing or asking for advice and/or confirmation of what he or she has done and the navigator providing feedback. The feedback can be pointing out simple errors, providing encouragement, or challenging the driver to explain what was done. An effective programming pair has an active relationship with both halves actively involved in the development of the code or design. Balance in the partnership is preserved by having the two switch roles on a regular basis.

Research results <sup>2-4</sup> indicate that pair programmers produce higher quality code in about half the time when compared with solo programmers. Some of this research was conducted at the

University of Utah in an undergraduate Software Engineering course<sup>2, 4, 5</sup>. In this work, the researchers observed a number of educational benefits, including increased satisfaction/reduced frustration from the students, increased confidence from the students on their project results, and reduced workload of the teaching staff. Similarly, educators at the University of California-Santa Cruz have reported on the use collaborative laboratory activities in an introductory undergraduate programming course, specifically in the form of pair programming<sup>6,7</sup>. They have found that pair programming improved retention rates and performance on programming assignments. The results of this work warranted further examination of paired programming in undergraduate instruction. Of particular interest was its impact in introductory computer science (CS1) courses taken by both computer science majors and engineering students. This paper reports on an ongoing study<sup>8</sup> at North Carolina State University (NCSU) during 2001-2002.

## **II. The Study**

The study was conducted in the CS1 course, Introduction to Computing – Java at NCSU. To date, this study has covered three semesters: Fall 2001, Spring 2002, and Fall 2002. The course is a service course with a majority of the students coming from the College of Engineering. While a majority of the students are freshman, they can be undergraduates of any level, graduate students, or continuing education students. This course is taught with two 50-minute lectures and one three-hour lab each week with students attending labs in groups of 24 with others in their own lecture section. In lab, students are given ‘completion’ assignments where they are to fill in the body of methods in a skeletal framework of a program prepared by the instructor. Students have to finish these programs in lab during the allotted time. In addition to the in-lab assignments, there were also three programming projects completed outside of lab. Students had to write complete programs for these projects.

Each semester the study was run, lecture sections were designated as either control (solo) or experimental (paired) sections. In some cases there were different faculty conducting the lectures for these sections, but they used a common syllabus. Each lecture section, in turn, had multiple lab sections that were taught traditionally (solo programming) for the control sections and using pair programming for the experimental sections. When students initially enrolled for the class, they had no knowledge of the experiment or that one section would have paired and other would have solo labs. All sections used the same lab exercises.

The pair programming arrangement raised the concern that there would be unequal contribution by each student towards the completion of the lab. Peer evaluations allowed students to rate their partners on a scale from 0 (poor) to 20 (superior) for each of five questions concerning contribution towards completion of the lab. Each student's lab score was multiplied by the percentage of points received on this evaluation. Though a majority of students received a 100% on their peer evaluation, low peer evaluation scores sent a strong signal to those who were not putting forth the necessary effort. Differences in ability and desire to contribute were also equalized by changing partners three times during the semester. Students were randomly assigned partners through a web-based program who they then worked with for two to three weeks. If a student's partner did not show up after 10 minutes, the student was assigned to another partner. If there were an odd number of students, three students worked together; no one worked alone. Finally, students were able to contribute as both driver and navigator by switching at regular intervals during the course of the lab.

Pairing was also done on the three projects done outside of class. There were concerns in the Fall 2001 semester that weaker students were being overly supported by stronger students. Because of this, starting in the Spring 2002 class, a policy was instituted that students must earn the right to pair program on the projects by attaining a score of 70% or better on the exams. This policy was then dropped in Fall 2002 due to student complaints; during this semester all students were required to pair with their lab partner.

### III. Findings

In the Fall 2001 and Fall 2002 semesters, the experiment was run in two sections of the course with the same instructor teaching both sections. In the Spring 2002 experiment, four sections were used with two different instructors. Our study was specifically aimed at the effects of pair programming on beginning students. Therefore, we chose to analyze the results of the freshman and sophomores only. We also only chose students who took the course for a grade, concluding that students who audited the class or took it for credit only had different motivations than other students. Table 1 reflects the number of students who were part of the analysis.

Table 1. Success Rate/Retention

Semester	Section	Number	C and above	Below C	Success Rate
Fall 01	Paired	44	30	14	68.18%
	Solo	69	31	38	44.93%
Spring 02 Instructor 1	Paired	82	54	28	65.85%
	Solo	76	50	26	65.79%
Spring 02 Instructor 2	Paired	198	113	85	57.07%
	Solo	26	15	11	57.69%
Fall 02	Paired	55	47	8	85.45%
	Solo	110	72	38	65.45%

### IV. Course Performance

For the Fall 2001, in the solo section, only 45% of the students we studied successfully completed the course with a grade of C or better. Comparatively, 68% of the students in the paired section met these criteria. A Chi-Square test revealed that this difference in success rates is statistically significant ( $\text{ChiSq}(1) = 7.056, p < 0.008$ ). These results are consistent with a similar study at the University of California UC-Santa Cruz that reported 92% of their paired class and 76% of their solo class took the final exam<sup>7</sup>. In the Spring 2002 semester, we observed no such difference in success rate. However, in Fall 2002 we again saw a significant difference in the success rate ( $\text{ChiSq}(1) = 7.292, p < 0.007$ ) with 85% of the paired students receiving a C or better while only 65% of the solo students did. Table 1 summarizes the results of the success/retention rate of students in the class across the semesters.

Midterm and final exam scores were also examined between the paired and solo sections for all three semesters. While some differences in scores were seen (with the paired sections out performing the solo sections), when using the SAT math (SAT-M) scores of the students as a covariate, an ANCOVA did not show any significant difference between sections with regards to midterm or final exam scores for any of the three semesters.

Table 2. Project Scores for the Fall 01 semester

Project	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

Using a similar analysis, an ANCOVA on the Fall 2001 data demonstrated a statistically significant difference in performance on Project 1 ( $F(1,94)=8.12$ ,  $p<0.0054$ ) and Project 2 ( $F(1,78)=4.52$ ,  $p<0.0367$ ) with the pairs performing better on both of them (Table 2). However this demonstrated improved performance does not occur in Project 3. In addition, these differences were not seen in the Spring 2002 semester, where the pair sections did not demonstrate correspondingly higher project scores. Analysis is currently underway for the Fall 2002 project scores.

## V. Attitudes

A survey was developed to measure general attitudes towards computer programming and computer science. This instrument was derived from the Fennema-Sherman mathematics attitudes scales<sup>9</sup>, modified to reflect programming and computer science rather than mathematics. The survey consists of a series of positive and negative statements. Participants respond to these statements on a five-point scale, ranging from strongly agree to strongly disagree. The negative statements are reverse coded prior to summing the subscale scores. The survey uses five of the seven subscale categories used in the Fennema-Sherman instrument: 1) Confidence in learning computer science and programming, 2) Attitude toward success in computer science, 3) Computer science as a male domain, 4) Usefulness of computer science and programming, and 5) Effective motivation in computer science and programming. The reliability of the new instrument was evaluated for internal consistency of the subscales with the responses from 162 students taking the CS1 course in the Fall 2001 semester<sup>8, 10</sup>. Values of Cronbach's alpha ranged from 0.83 and 0.91 for the five subscales.

When given at the end of the Fall 2001 semester, no significant differences on any of the five subscales were seen between the pair and solo sections. In subsequent semesters, a more general question about attitudes about pair programming was asked: *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?* As seen in Table 3, 80% of the students in the Spring 02 semester and 85% of the students in the Fall 02 semester expressed no misgivings about working in pairs.

Table 3. Response to the question about willingness to pair program again.

Semester	Number of Respondents	Yes	No	I don't care
Spring 02	207	124 (59.90 %)	42 (20.28%)	41(19.80 %)
Fall 02	71	46 (64.77%)	11(15.49%)	14 (19.71 %)

## VI. Lab Observations

Below is a summary of weekly observations done in both paired and solo labs over the course of the Spring 02 and Fall 02 semesters<sup>11</sup>. Different authors conducted the observations in these two semesters.

### Paired Labs

While paired students differed markedly in their approach to the pair-programming protocol, common behavioral patterns seemed to emerge during the paired lab sessions. In every lab session, at least one pair of students followed the pair-programming protocol perfectly—taking on the appropriate roles (driver or navigator), switching roles when they were told to, discussing and helping each other. In these pairs, the navigators were very attentive to the driver even when no dialogue was taking place. These individuals were generally in close proximity and only one computer would be used. These individuals would stand up and switch seats usually 2-3 times during a three hour lab.

A majority of pairs deviated from this “perfect” protocol in various ways. The most common variation involved role-taking issues. With the exception of a few pairs, most student pairs seemed reluctant to reverse roles immediately after being told to do so by the instructor. Some students remained in the same role—either navigator or driver—during the entire lab session. Other students reversed roles at times other than those indicated by the lab instructor or moved between pair and independent work. For example, pairs would both work independently for the first hour and then it appeared that one would get stuck and then join the other and assume the navigator role. In either case, it seemed that students either found it inconvenient or unnecessary to take on assigned roles. There were a few pairs, all male, where it was harder to determine if they were working as driver/navigators or not. These pairs would never get in close proximity but you could generally observe that one was typing and the other was reading from a distance and would assist when necessary.

When instructors explained and reinforced the pair programming protocol on a regular basis, students were more apt to assume appropriate roles as well as reverse roles when necessary. In labs where instructors forgot to ask students to reverse roles, no role reversal occurred. In labs where the instructor failed to enforce the pair-programming protocol, students often opted for individual work.

Whether or not students take on their appropriate roles during the specified times, they did show increasing willingness to take on the driver/navigator roles with each passing week. Easing into the pair-programming protocol over time may suggest that students need time to become familiar with the paired protocol before they can feel comfortable. This makes sense since undergraduate students may not be accustomed to a collaborative learning approach in a computer lab session.

In general, students in the pair-programming lab sessions showed a high level of interaction with each other. Students were discussing issues related to the programming assignment on a consistent basis. Students questioned, directed, and guided each other throughout the lab session. When student pairs could not seem to answer questions on their own, they would ask the instructor; but the interaction with the instructor was usually very brief (less than five minutes). Overall the paired labs seemed to be much more efficient than the unpaired labs. Although there was a lot of dialogue during the class it never seemed chaotic, as students seemed to stay focused

on their tasks. The paired labs provided structure to the communication between students, which made it more effective than the unpaired labs.

A lot of students' questions to instructors were of a logistical rather than conceptual nature. On a very frequent basis, pairs resolved their own problems without the instructor's help. Overall, instructors spent very little time answering questions. Most instructor-student interactions seemed to take the form of extended discussions. Students would want to know how to apply what they were doing to another scenario. Hypothetical discussions of applications showed evidence of higher-level thinking processes that went beyond the scope of the programming assignment.

### Solo labs

Unpaired labs were quieter than paired labs, although there was certainly a fair amount of interaction in the unpaired labs. Typically there were a number of different types of interactions in unpaired labs. A few students would wear headphones during lab and they typically stayed very focused on their tasks and did not appear to ask as many questions as their classmates. These individuals worked independently and were very quiet. In comparison, other students were highly interactive with one another. There were two different types of inter-student interaction that was typically seen. The most common type was when several people sitting on the same row would converse both on and off topic during the lab. These students often asked each other questions especially if the TA was busy. This interaction is different from the paired arrangement in that the students did not navigate for each other. Occasionally one student would point to his/her code and then the other student would point to their code and they would compare/discuss. This type of interaction was much more random and unstructured compared to what was observed in the paired labs. Here, there was typically no cross talk between rows.

A second type of inter-student interaction observed was when a student would essentially ask anyone who was available for help. These students were like scavengers. Their eyes would look on anyone's monitor they could see (beside and behind them). They would often look around and would ask a question to anyone who made eye contact with them regardless of where they were sitting (often talking over rows). These students were constantly looking around, they never seemed focused on their own work. Occasionally they would get up and go to the computer of their helper and vice versa. The instructors were often so busy with questions that they didn't seem to mind.

Another type of student observed was that of the student who monopolizes the instructors time. In some cases they would ask question after question. In other cases they would ask questions that required very lengthy responses and in some cases want instruction on skills the instructor would tell them they should already possess. Usually there were only one or two of these students in a class and in some labs there were none.

When the instructors answered students' questions, they spent a minimum of five minutes and a maximum of twenty minutes with each student. Instructors remained busy answering questions for the duration of the lab sessions, and at times seemed overwhelmed. Students needing assistance and who were more resourceful would ask a fellow classmate a question and continue working. Often however, students with questions sat and waited for long periods of time (maximum of thirty minutes) before they could get help. During this time, students seemed "stuck" and could not go any further. Some students in these situations opted to help each other,

but their interactions remained brief and sporadic. In many cases, if several students were all waiting for the TA they would start chatting with each other off topic. On some occasions when they needed help but their neighbor seemed too busy to help them, students leaned over to look at their neighbor's computer screen.

In an attempt to quantify the differences in Student/Instructor interactions between the pair and solo labs, counts were made in Fall 2002 of the total number of questions asked and the number of times the students gave up trying to get a question answered (Table 4). A total of N=9 pair lab sessions and N=7 solo lab sessions were used in this analysis. Using a Student's t-test, no significant difference was seen between the pair and solo sections for either measure. However, the descriptive statistics do show a general trend of fewer questions and fewer 'Give Ups' for the pair section. There is also smaller variance for both measures for the pair section.

Table 4. Question asking during the lab sessions. 'Give Ups' are the total number of times students gave up getting their question answered.

Section	Total Questions Mean	Total Questions Standard Deviation	No. of 'Give Ups' Mean	No. of 'Give Ups' Standard Deviation
Paired	43.11	7.47	1.14	0.90
Solo	56.14	30.68	3.17	2.56

## VII. Pair Section Focus groups

In the Spring 2002 semester, separate focus groups were run for both students and lab instructors in the pair sections. In the student focus group, eight major themes were identified: Pairing and Learning, Partnerships' Negative Dynamics, Communication, Mixed Abilities, Background & Personality, Forming a Partnership, Pairing & Real World Scenarios, Responsibility & Accountability, and Switching Roles.

Students stressed the advantages of pairing throughout the focus group discussion. Primarily, students brought up issues about being able to ask questions immediately as problems arise rather than having to wait for an instructor. Having someone there while working on problems seemed to help students clarify ideas, pick up on minor errors, and work on understanding conceptual knowledge. Students mentioned that pairing was particularly beneficial when working on projects, because two people often complement each other by sharing strengths and filling weaknesses. Project partnerships came up in the discussion as an important aspect of learning especially for students that are lagging behind despite continued efforts.

Students also emphasized that the secret to the success of pairing is finding someone that feels comfortable with pairing. When students work well together, the partnerships seem to be successful, even if a low-ability student is paired with a high-ability student. They claimed that students who have no interest in learning do poorly or fail regardless of classroom arrangements. Students also found that the paired setting mimics real world settings where people are often randomly matched to work together on programming projects.

The comments that the Lab Instructors gave during the course of the focus group revealed thirteen major themes: Pairing Protocol, Time & Efficiency, Student Interaction, Higher Order Thinking & Active Learning, Unpaired Labs, Reluctance to Pair/Switch Roles, Motivation,

Student Compatibility, Pairing & Learning, Pairing Protocol Logistics, Lab Instructor Responsibility & Evaluation, Project Pairing, Confidence, and Future Success.

The Lab Instructors all agreed that implementing the paired protocol gave them flexibility and time that they could use to give students equal opportunities for questions, discussions, and other support. The Instructors also found their jobs fun and easy when teaching in paired labs. They observed that students interacted positively for the most part, and that partnerships helped students reduce major problems associated with programming.

Some instructors noted that students would benefit immensely from pairing during projects and were puzzled by students who opted not to pair. Other instructors felt that projects should give students an opportunity to work alone so that they can feel confident when programming on their own. Overall, instructors found the paired protocol to be very successful and worth considering as an alternative to traditional ways of teaching CS1 laboratory sections.

In both focus groups, the students and instructors noted the importance of having “compatible” partners. Two suggestions for constructing compatible pairings were to have them be based on personality type and/or on skill level.

### **VIII. Conclusions**

Perhaps not surprisingly, the main impact of pair programming on the CS1 course is seen in the lab. Here the impact has been seen, first and foremost, on the instructional environment but also on actual performance in the class. The impact on the instructional environment affects how problem-solving takes place while the students are working on their programming assignments and on how lab instructors are able to allocate their resources to facilitate this activity. The impact on actual performance in the class is less direct. Some differences can be seen in how students perform on their project assignments done in conjunction with the labs. There is reason to believe that the improvement in the ‘quality of life’ in the pair programming labs may also contribute to a higher overall success rate in the course.

A comparison of exam scores between the pair and solo sections did not reveal a significant difference when the student’s SAT-M scores were used as a covariate. From this we might conclude that the students in the paired sections perform at least as well as those of the solo sections, indicative of their ability to demonstrate mastery of the subject material when they must work together with a partner in a lab setting. Another explanation for this lack of significant difference may be the less than direct connection between the content on the exams and work done in the labs. While the lab work is meant to reinforce lecture topics and the material found on the exams, the exams were primarily based on material covered in lecture. A closer examination of the instructional connection between lecture, exam, and lab content might reveal the validity of this conjecture.

Another possible hypothesis centers around the dynamics of the student ability distribution over the semester. The higher success rate for the two Fall semesters meant that more students were able to keep their overall grades at a C or above at the end of the semester. However, it does not provide a more fine-grained analysis of distribution of grades, nor does it indicate how many students withdrew over the course of the semester in the two sections. The significantly higher success rate for the pair sections in two of three semesters points to some element of treatment

providing additional support for the students. It is possible that pair programming in the labs provided enough of an attitudinal boost to motivate weaker students to continue working on passing the class when they might otherwise give up. Weaker students who stayed in the pair class might have kept over the C wall but still weighted the paired section towards an overall lower performing group than the solo section.

This might also explain why there was no significant difference in success rate for the Spring 2002 semester. The Spring semester is when CS majors normally take the course and it would follow that this population more intrinsically motivated to complete this course. This hypothesis of a higher withdraw rate of weaker students in the solo section may also explain the lack of difference in performance in Project 3 in the Fall 2001 semester. If this is true, then more of these lower performing students in the solo section would have dropped prior to completing Project 3. A more detailed analysis of performance by major and a more fine-grained analyses of grade distributions may shed more light on these hypotheses.

The indirect connection between paired programming and more general attitudes surrounding computer programming and computer science may explain why a single course using paired programming did not significantly alter students' outlooks relative to the control solo section in the Fall 2001 survey. However, the more direct question asked the Spring and Fall of 2002 garnering students' feedback on pair programming did provide an indication that pair programming in the CS1 lab, in and of itself, was not a negative experience.

The in-lab observations and focus groups provided a rich set of data to demonstrate some of the key strengths of pair programming. Pair programming at its best provided a framework in which students could optimize their efforts in designing and solving programming problems. Partners provided immediate feedback and motivation during the programming process. They also provided an immediate first avenue for answering questions and solving problems while working on the program. It also created an overall lab environment that kept attention focused on the problem at hand.

Our lab observations also indicated that pair programming was working at its best for probably only a plurality of pairs in the labs. While the existence of less than optimal group dynamics could be readily observed, the causes behind this behavior are not so clear. It is probably safe to say that the students in the class had spent a majority of their secondary and post-secondary academic life engaged in solo activities. If they did participate in group projects, they were probably more likely to be of a cooperative nature (i.e., the work was divided among the members to be done individually) rather than of a collaborative nature. In addition, academic programming is one of the less likely activities to be done as a group endeavor. All of this points to a lack of experience with, probable questioning of the efficacy of, and the inevitable concern over pair programming's effect on their grade. All of these may be contributing factors to the observed resistance on the part of some to fully participate. It is worth noting, however, that a general trend towards better compliance was seen over the course of the semester.

Lack of compliance with role reversal may also have roots in more immediate issues. Perhaps reversing roles at prescribed time interrupts the flow of work, so students opted not to reverse roles; or students need to find a pausing point before they can reverse roles. In the Fall 2002 semester, role reversal was flagged at breakpoints within the exercise rather than having the instructor call for a switch after a set number of minutes. This took advantage of more natural break points and eased the load on the instructor.

Another reason for problems with role reversal or other pair dynamics might be found in social mismatches. Students may be mismatched based on their achievement level, gender, or cultural roles<sup>12</sup>. In such cases, it is common for one student in the group to take over. For example, it could be that in the pairs there was a dominant individual who was also a superior programmer who was not willing to play their defined role or otherwise cooperate with their partners. Students in the Fall 2002 semester experimental section were administered the Meyers-Briggs Personality Type Indicator<sup>13</sup>. Based on their scoring they were partnered with students of their same personality type, opposite type, or randomly for different thirds of the semester. Analysis of this data is ongoing.

Regardless of their misgivings about pair programming, some students did realize that this approach did mimic what they were likely to face in the workplace after graduation<sup>14</sup>. In the workplace, students would not necessarily have the option to choose whether they were going to work alone and almost certainly would not be able to pick partners based on personality type.

The pair programming experiment has also been a learning experience for both the faculty members teaching the course and the instructors – undergraduate TA's – teaching the labs. Closed labs are excellent for controlled use of pair programming<sup>6</sup>. The instructor can ensure that people are, indeed, working in pairs at one computer and ensure that the roles of driver and navigator are rotated periodically. However, many classes have programs that require work outside of a closed lab and it would be difficult to enforce pair programming in this case. Some students will come to fully appreciate the benefits of pair programming and will seek to work with a partner outside of class. Others will choose to work alone, whether they prefer pair programming or not, often because they would prefer to work on homework in the comfort of their dorm room at any hour of the day or night. Pairing outside of lab requires effort in planning and coordination; some students view the added coordination as a hassle.

Even with the closed lab, full cooperation of the lab instructor is critical. Without instructor reinforcement, students very easily reverted to the individual work with which they are so accustomed. Lab observations certainly indicated that lab instructors who did not take the pair programming protocol seriously had a much higher level of non-conformance in their labs. Most instructors in the pair sections did come to realize the personal benefits to the instructional approach. Many had previously taught traditional lab sections (or were teaching them the same semester) and noted the differences in lab management. They observed self-management of groups and the way they were often able to overcome their own programming difficulties. The small amount of data on question asking collected in the Fall 2002 semester seemed to support these impressions. Hopefully further data can be collected to bolster this finding.

### **Acknowledgements**

The National Science Foundation Grant DUE CCLI 0088178 provided funding for the research in this pair programming project.

## **Bibliography**

1. Beck K. Extreme programming explained: Embrace change. Reading, MA: Addison-Wesley; 2000.
2. Cockburn A, Williams L. The costs and benefits of pair programming. In: XP2000 Cagliari, Sardinia, Italy: Addison-Wesley; 2000.
3. Williams L, Kessler RR, Cunningham W, Jeffries R. Strengthening the case for pair programming. IEEE Software 2000;July/August:19-25.
4. Williams LA. The Collaborative Software Process [Ph.D. Dissertation]: University of Utah; 2000.
5. Williams L, Kessler RR. Experimenting with industry's "pair-programming" model in the computer science classroom. Journal of Computer Science Education 2001;March.
6. Bevan J, Werner L, McDowell C, . Guidelines for the Use of Pair Programming in a Freshman Programming Class. In: Fifteenth Conference on Software Engineering Education and Training (CSEE&T 2002); 2002; 2002.
7. McDowell C, Werner L, Bullock H, Fernald J. The Effects of Pair Programming on Performance in an Introductory Programming Course. In: ACM, editor. Proceedings of the Conference of the Special Interest Group of Computer Science Educators (SIGCSE 2002). New York, NY: ACM; 2002.
8. Williams L, Wiebe E, K.Yang, Ferzli M, Miller C. In support of paired programming in the introductory computer science course. Computer Science Education 2002;12(3):197-212.
9. Fennema E, Sherman JA. Fennema-Sherman mathematics attitudes scales. Instruments designed to measure attitudes toward the learning of mathematics by females and males. JSAS: Catalog of Selected Documents in Psychology 1976;6(31):(Ms. No. 1225).
10. Wiebe EN, Williams L, Yang K, Miller C. Computer Science Attitude Survey. Technical Report: Dept. of Computer Science, North Carolina State University; 2003 January 10. Report No.: TR-2003-01.
11. Ferzli M, Wiebe EN, Williams L. Paired Programming Project: Focus Groups with Teaching Assistants and Students. Technical Report: Dept. of Computer Science, North Carolina State University; 2002 November 25. Report No.: TR-2002-16.
12. Webb NM, Nemer KM, Chizhik AW. Equity issues in collaborative group assessment: Group composition and performance. American Educational Research Journal 1998;35(4):607-651.
13. Bishop-Clark C, Wheeler DD. The Myers-Briggs personality type and its relationship to computer programming. Journal of Research on Computing in Education 1994;26:358-70.
14. Collings P, Walker D. Equity issues in computer-based collaboration: Looking beyond surface indicators. In: Schnase JL, Cunniss EL, editors. Proceedings of CSCL '95: The First International Conference on Computer Collaborative Learning. Mahwah, NJ: Lawrence Erlbaum Associates; 1995. p. 75-83.

## **Biography**

ERIC N. WIEBE, Ph.D.

Dr. Wiebe is an Assistant Professor in the Department of Mathematics, Science, and Technology Education at NC State University. He received his Doctorate in Psychology and has focused much of his research on issues related to technology in the instructional environment. During the past nine years, he has worked on the integration of scientific visualization concepts and techniques into both secondary and post-secondary education. Dr. Wiebe has been a member of ASEE since 1989.