In Support of Pair Programming in the Introductory Computer Science Course

Laurie Williams, Eric Wiebe, Kai Yang, Miriam Ferzli, Carol Miller
North Carolina State University
{lawilli3, wiebe, kyang, mgferzli, miller}@unity.ncsu.edu

Abstract

A formal pair programming experiment was run at North Carolina to empirically assess the educational efficacy of the technique in a CS1 course. Results indicate that students who practice pair programming perform better on programming projects and are more likely to succeed by completing the class with a C or better. Student pairs are more self-sufficient which reduces their reliance on the teaching staff. Qualitatively, paired students demonstrate higher-order thinking skills than students who worked alone. These results are supportive of pair programming as a collaborative learning technique.

In industry, programmers collaborate for the majority of their day. In *Peopleware* (DeMarco and Lister, 1987), it was reported that 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. Yet, most often when completing their degree, programmers must learn to program alone; collaboration is considered cheating. This is unfortunate not only because collaboration is encouraged and required in a student's future professional life, but there are also findings that cooperative and collaborative pedagogies are beneficial to students (Slavin 1980, 1990).

An emerging software development methodology, Extreme Programming (XP) (Beck, 2000), has recently 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*, types at the computer or writes down a design. The other partner, called the *navigator*, has many jobs. One is to observe the work of the driver – looking for defects in the work of the driver. The navigator has a much more objective point of view and is the strategic, long-range thinker. Additionally, the driver and the navigator can brainstorm on-demand at any time. An effective pair programming relationship is very active. The driver and the navigator communicate, if only through utterances, at least every 45 to 60 seconds. Periodically, it's also very important to switch roles between the driver and the navigator.

Research results (Williams, Kessler et al., 2001; Cockburn & Williams, 2002) indicate that pair programmers produce higher quality code in about half the time when compared with solo programmers. These results were based on experiments held at the University of Utah in the senior-level Software Engineering course (Williams, 2000; Williams & Kessler, 2000; Williams & Kessler 2001). The focus of this research was on the affordability of the practice of pair

programming, 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 of the teaching staff.

These observations inspired further research directed at the use of pair programming in educating Computer Science students. This paper details the results of an experiment that was held at North Carolina State University (NCSU) in 2001. The experiment was specifically designed to assess the efficacy of pair programming in an introductory classroom.

Background

In an instructional setting, how the efficacy of pair programming is evaluated shifts somewhat from the industrial setting. For example, while industry is very concerned with issues of "efficiency" (i.e., how many worker hours are needed to complete a block of code), educators are more concerned with learning outcomes, attitudes of the students and instructors, and classroom management issues. Taking pair programming to an instructional setting, it is important to correlate how pair programming is defined in industry with current pedagogical techniques being researched and applied in instructional settings. According to Jehng (1997), approaches to peer-based interaction in the classroom can take three primary forms: 1) tutoring, where the less capable are guided by the more capable; 2) cooperation, where learners work on different parts of the task; and 3) collaboration, where learners work jointly on almost all parts of the task. Using this definition, the goal of pair programming in an instructional setting is collaboration, with the realization that tutoring will occur between pairs that are not evenly matched. However, it is the goal of pair programming to avoid a cooperative type interaction. Jehng concludes that collaboration typically offers richer social interactions and is more productive, but individuals can dominate.

With this focus, it is worth reviewing a number of studies, which have relevance to the current research. In summarizing recent research in science education, Baker & Palmer (1997) concluded that in the secondary school science classroom, in-depth interaction between students is central to the process of constructing meaning of science topics and solving science-based problems using this knowledge. While scientific knowledge can be constructed by individuals in isolation, group work forces the individual to articulate and defend their postulations and solutions. Group interaction provides multiple perspectives when reflecting potential hypotheses and interpretations of results. Roth (1993) notes that this social constructivist perspective supports the use of collaborative activity where individuals in the group are not equal; that even in this environment all concerned can continue to grow in their knowledge. He goes on to note that collaborative activities in the classroom provide valuable experience in cognitive apprenticeship and social enculturation whereby students have the opportunity to develop social interaction skills necessary for success after leaving school.

The positive aspects of collaborative activities in the classroom have also been seen in university mathematics courses. Yusof & Tall (1998) report that the use of collaborative problem-solving activities improved attitudes of students towards mathematics. He goes on to discuss that one of the biggest challenges is working on how to effectively integrate collaborative problem-solving activities into the courses. Abboud (1994) concludes that problem-solving activities are compatible with a collaborative classroom structure. As

implemented by Abboud, this type of classroom is constructivist in nature and helps promote a process orientation to program design.

The experience of Abboud points to a connection between instructional approach and programming abilities. In a comparison between a traditional lecture and a laboratory-based programming course, Romeu & Alemzadeh (1998) demonstrated significant improvement in program development performance and pop quiz scores for those students in the laboratory course. While not specifically using collaborative learning techniques, Romeu and Alemzadeh demonstrated that changes in instructional approach can influence performance in a programming course. Similarly, Oliver & Malone (1993) found that the level of conceptual, lab-based activity was a significant factor in improving higher order thinking skills in programming, including semantic programming knowledge. They went on to cite earlier work supporting the positive influence of group activity in the laboratory.

MacGregor (1988) investigated an instructional approach that emphasized both collaborative techniques and structured design concepts. He found that this approach led to improved ability in designing programs, overall programming performance, and improved student attitudes towards programming. He also noted advantages for the instructors; students working in groups tended to resolve questions via their peers rather than rely exclusively on their instructors. In addition to immediate gains in programming ability, MacGregor pointed out the importance in modeling the team-oriented approach many businesses employ in the classroom. Priebe (1997) reiterated these points, emphasizing how team-oriented activities in the classroom modeled real-world teamwork in industry. Priebe believed that the group setting fostered a positive peer pressure that led to neater and more complete assignments being handed in. He, too, commented on the higher level of self-teaching that occurred in the cooperative group setting.

More recently, 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 (Bevan, Werner, et al., 2002; McDowell, Werner, et al., 2002). They have found that pair programming improved retention rates and performance on programming assignments.

Research by Lips & Temple (1990) has suggested that interest and enjoyment, previous experience in computer science, and mathematical ability all play a role in deciding to major in computer science. Having identified similar key factors, Smith (1994) tracked both affect and achievement when investigating differing instructional methods in a programming course. She also used the SAT-M (mathematics) scores of the students as a covariate to account for the differing mathematical abilities of study participants. Achievement measures may encompass both activities directly related to the collaborative lab activities and those more closely associated with other aspects of the course. Jehng (1997) believes that collaborative activities' impact on knowledge production may reflect itself in a number of ways in a computer science course.

In summary, pair programming activities in a lab portion of a computer programming course may influence both achievement and attitudes of the students. Achievement measures may include assessments done both in the lecture and laboratory components of a course. In addition, achievement may be indirectly measured through the number of students passing the course versus those receiving a D, F, or withdrawing from the course during the semester (i.e., retention rate). Effective changes in students may be assessed by looking at their confidence and motivation in learning computer programming, their perceived likelihood of success, and

the likelihood of future use of programming skills. Finally, the success of any change in instructional strategies depends on adoption of the techniques by instructors. For that reason, it is also important to look at perceptions of the efficacy of the techniques held by instructors. These factors include ease of administration, quality of the laboratory experience and academic integrity issues.

Experiment

The fall 2001 semester marked the beginning of an NSF-supported longitudinal study of the efficacy of pair programming to improve student learning, success, and retention. (Data will be collected on the students in this study for a two year period to examine the longer-term effects of pair programming.) An experiment was conducted in the CS1 course, Introduction to Computing – Java. The course is taught with two 50-minutes 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, whereby students are given a weekly assignment to complete during the allotted time. The 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 three programming projects that are completed outside of the closed lab. The projects are generative, in that students start from scratch without any structure imposed by the instructor. The course is a service course, and is therefore taken by many students throughout the university, though most students are from the College of Engineering. Additionally, most students are freshman; however students of all undergraduate and graduate levels also take the course to learn competitive programming skills.

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 in both sections. (The exams were given to the second section immediately after the first, leaving little time for students to tell the second section about the exam content.) One section had traditional, solo programming labs. In the other section, students were required to complete their lab assignments utilizing the pair programming practice. When students enrolled for the class, they had no knowledge of the experiment or of that one section would have paired and other would have solo labs. In the pair programming labs, students were randomly assigned partners based on a web-based computer program and not student preferences. They worked with the same partner 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.

In the Fall, 112 students were in the solo section and 87 were in the paired section. 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 were not as motivated to excel as other students. This reduced our sample size to N=69 in the solo section and N=44 in the paired section. In our experiment, we examined the following hypotheses:

- A higher percentage of students that have participated in pair programming in CS1 will succeed in the class by completing the class with a grade of C or better.
- Students' participation in pair-programming in CS1 will lead to better performance on examinations in that class

- Students' participation in pair-programming in CS1 will lead to better performance on course projects in that class
- Students' participation in pair-programming in CS1 will lead to a more positive attitude toward the course and toward Computer Science in general
- Students' participation in pair-programming will lead to a lower workload for course staff

We had hoped to also separately study these results for the women and the African-Americans in the class. However, there were far too few women (12 in solo, 4 in paired) and African American (8 in solo, 6 in paired) to allow for statistically evaluation. Statistics on women and minorities will need to be tracked over several semesters to yield meaningful results.

As educators, we were concerned that the academically weaker or less motivated students would compel their partner to do all the work. To alleviate this concern, each time the students were assigned a new partner, they were required to complete a peer evaluation on their prior partner. The students rated their partner on a scale from 0 (poor) to 20 (superior) for each of these five questions:

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

The sum of the ratings on each of these questions yielded a grade from 0-100%. Each student's lab grade was multiplied by this peer evaluation factor. For example, if a student had a 90% lab average but a peer evaluation score of 50%, they received a final lab grade of 45%. We had successfully used this form of peer evaluation in past classes. We have found that it does motivate students to do their share of the work. In general, 95% of the class will report that their partner did their share of the work, and would assign him or her a 100%. In a minority of cases, the peer evaluation score is a strong signal of a student who is truly not putting forth the necessary effort.

Closed labs are excellent for controlled use of pair programming (Bevan, 2002). 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. In the CS1 experimental course, the students completed three programming projects outside of the closed lab environment. We gave the students in both sections the option of working alone or pair programming for these projects. Many chose to pair program. However, we found instances of students who were doing very poorly in the class pairing with students who were dong well in the class. Often, these students did well on the projects, causing suspicion that the stronger student did most or all of the work. As a result, starting in the Spring 2002 class, we instituted a policy that students must earn the right to pair program on the projects by attaining a score of 70% or better on the exams.

Quantitative Findings

Success Rate/Retention

We examined the percentage of students who succeeded in the class. Our criteria for "success" was the completion of the course with a grade of C or better. Historically, beginning Computer Science classes have poor success rates. For all the good intentions and diligent work of computer science educators, students find introductory computer science courses very daunting—so daunting that typically one-quarter of the students drop out of the classes and many others perform poorly.

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 this criteria. A Chi-Square test reveled that the difference in success rates is statistically significant ($?^2(1)=7.056$, $p < 0.008$). This result is consistent with a similar study at the University of California UC-Santa Cruz (McDowell, Werner, et al., 2002).

Performance on Examinations

On average, students in the paired section performed better on the two midterm examinations and the final examination, as shown in Table 1. We removed any scores of 0 from our analysis; this decision is based on the fact that students who attempted to take the exam all received a grade higher than 0.

Table 1: Exam Scores

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

An analysis of SAT-M scores revealed that the students were not uniformly distributed across groups. The students in the paired group had a mean SAT-M score of 662.10 while the solo group had a mean score of 625.43. The One-Way ANOVA revealed that this difference was statistically significant ($F(1.101)=5.19$, $p<0.018$). An ANCOVA further revealed a correlation between SAT-M scores and exam scores, when considered for each exam individually ($F(1,98)=36.32$, $p< 0.0001$; $F(1,98)=41.41$, $p<0.0001$). When using SAT-M as a covariate, an ANCOVA does not show any significant difference between sections with regards to midterm 1 and 2 or final exam scores. Based on these results, we cannot conclude that pair programming in the laboratory helped students perform better on exams.

We wish to discuss two factors that may influence these results. First, changes in the lab portion of the course may have enough of an influence on student work and attitudes to keep them from dropping out of the course or boosting their grades enough to pass the course. This may have dampened the overall distribution of grades in the paired section since it may have kept poorer performing students in the calculation pool. Researchers at UC-Santa Cruz have made this same speculation (McDowell, Werner, et al., 2002) when reflecting on results. Additionally, only approximately 40% of the exam content required program code to be written in the answers. The rest of the exams were short answer and multiple choice. Quite feasibly, pair programming might not help students answer short answer and multiple choice questions on the material better. The classes' scores on only the portion of the exam that did require programming are not available.

Performance of Programming Projects

On average, students in the paired section performed better on the two of three programming projects, as shown in Table 2.

Table 2: Programming Project Scores

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

The ANCOVA demonstrated a statistically significant improvement in performance of the pairs on Project 1 ($F(1,94)=8.12$, $p<0.0054$) and Project 2 ($F(1,78)=4.52$, $p<0.0367$). However this demonstrated improved performance does not occur in Project 3. Perhaps, this is because by Project 3 the lower performing students had dropped in the solo section but were still working in the paired section.

Attitude

We hypothesized that students in paired labs will have a more positive attitude toward the course and about Computer Science in general. We based this hypothesis on prior observations that beginning classes can be very frustrating. Students might debug a program for several hours because of a very simple syntactical error. These kinds of errors would likely be caught by the navigator, precluding the need for extensive, frustrating debugging sessions.

We evaluate this hypothesis in two ways. First, we administered an extensive survey to students at the beginning and end of the semester. The survey was developed to measure attitudes towards computer programming and computer science in general. This instrument was derived from the Fennema-Sherman mathematics attitudes scales (Fennema, 1976), modified to reflect programming and computer science rather than mathematics. This survey consisted of a series of subscales measuring: (1) self-confidence (2) motivation, (3) attitudes toward success, (4) females in Computer Science, and (5) usefulness of Computer Science. The reliability of the new instrument was evaluated for internal consistency of the subscales with the responses from 162 students taking the introductory computer science course. Values of Cronbach's alpha ranged from 0.82 and 0.91 for the five subscales. A non-parametric analysis of variance (Kruskal-Wallis) indicated that there was no significant difference any of these indices between the solo and pair sections based on the end of the semester measurement. These findings do not support our hypothesis.

We also looked at specific questions on the Computer Science department student course evaluation, because both sections were taught by the same instructor. The only data available on course evaluation is a mean score, so no statistical evaluation could be performed. On the course evaluation, a 1 is an unfavorable score and a 5 is a very favorable score. As shown in Table 3, students did feel more favorable toward the course and the instructor in the paired section:

Table 3: Course Evaluations

Exam	Paired	Solo

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

Qualitative Findings

In order to gain insights about the student-student, instructor-student dynamics of the pair-programming protocol, we collected observations during paired and solo computer programming laboratory sessions. Observational data was collected in the Spring semester 2002 on a weekly basis, during a continuation of the controlled study conducted in the Fall semester 2001. The setting for the observations was the CS1 laboratory, which students attend for three hours every week. Analysis of the observational data allowed us to document major issues related to the pair-programming protocol that would not surface during the other components of the study.

Students' Behaviors during Pair-Programming

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. Every other type of interaction during the pair-programming protocol deviated from this "perfect" scenario 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. 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. In either case, it seemed that students either found it inconvenient or unnecessary to reverse roles. Perhaps reversing roles at prescribed times 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.

Whether or not students take on their appropriate roles during the specified times, they do 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. Most students in this type of setting are used to individual work, even though they will encounter a collaborative approach to programming and project building in the workplace (Collings & Walker, 1995).

Although the pair-programming protocol is set up to give students an opportunity to experience two different roles while programming, some students remain marginal players in this setting. Several students were observed to give little or no input during the entire lab session. According to cooperative/collaborative learning research, there are several reasons why students remain disengaged. Students may be mismatched based on their achievement level, gender, or cultural roles (Webb, Nemer, Chizhik, 1998). In such cases, it is common for one student in the group to take over.

Instructors' Roles in Pair-Programming

The role of the laboratory instructor seems crucial to the success of the pair-programming protocol. 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 opted for individual work. Students who chose to pair on their own did not follow the correct protocol. These students worked at their own computers while they engaged in some level of collaboration with no evidence of driver/navigator roles. Instructors that did enforce the pair-programming protocol, were more likely get students involved in team learning. Without instructor reinforcement, students very easily reverted to the individual work with which they are so accustomed.

Pair-Programming and Learning

Without exception, 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). 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

Overall, the solo labs were very quiet. There was little or no discussion between students, and students had questions on a frequent basis. 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. Often, 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. 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.

Summary of Observational Findings:

Students in paired labs engage in extensive discussion throughout the entire lab session, and students seem to help each other resolve questions. Instructors spend more time discussing advanced issues with students, rather than answering basic questions. Students show evidence of higher order thinking—synthesizing and applying lab material to other scenarios. Students seem to take on navigator/driver roles with more ease when these roles are reinforced by the instructor. Students switch roles at their own pace, perhaps because they need to reach an agreed "cut-off" point.

Students in solo labs who run into problems spend a lot of time waiting for the instructor to help them. Although students seem to interact with each other, interactions are brief. Discussion between students is challenging, because students are at different points in their work and

discussion would disturb their progress. Instructors have to take on a very active role when helping students, rather than letting students figure things out on their own.

Summary and Future Work

Previous anecdotal evidence supported 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 of the teaching staff. A structured, empirical study began at NCSU in the Fall 2001 semester to qualitatively and quantitatively examine these anecdotal claims in a CS1 course; this paper outlines the results of one semester of this study. Our empirical results indicate that students who practice a pair programming technique in closed labs are more likely to persevere through CS1 and receive a grade of C or better. Student pairs also produced better grades on programming assignments. The students who are in the paired labs received higher grades, on average, on examinations, though this difference was not statistically significant. Qualitative observations supported that paired closed labs were a superior learning environment for the students. Paired labs are also less stressful for the teaching assistants because students are not as reliant on them as the sole provider of technical information and help.

What are the implications of these research results to Computer Science departments? Our findings support that CS1 closed labs would benefit from transitioning to paired labs. However, our results also caution that with the benefits realized with student pair programming come some costs. Teaching assistants must enforce the pair programming model by reminding students to periodically rotate the driver and navigator roles and to notice when one student might be dominating. In order to create an environment where all students have the opportunity and the incentive to participate in assignments, we recommend that partners are assigned and are not static for the entire semester. Also, students should be provided the opportunity to report via a peer evaluation on the contributions of their partners; these peer evaluations should have significant weight in a student's grade. Lastly, we advise that lower performing students are not given the opportunity to pair on assignments done outside of a closed lab.

In the future, we will examine the success rate of these students, as measured by their future grades in Computer Science courses, as they continue with their academic career. Additionally, we will accumulate results of women and African-Americans to examine if pair programming can aid in their success. We also believe that pairs are not only better at successfully completing a program and/or programming project, but they also develop superior designs and utilize more sophisticated programming techniques. To validate this hypothesis, we plan to examine the structure of the code produced by solo programmers vs. pairs.

Acknowledgements

Funding for the research was provided by the National Science Foundation (CCLI 29728000). We also thank our reviewers, in particular Dr. Rich Felder, for helping us improve this journal article.

Bibliography

Abboud, M. C. (1994). Problem solving and program design: a pedagogical approach. Computer Science Education, 5, 63-83.

Baker, D. R., & Piburn, M. D. (1997). Constructing science in middle and secondary school classrooms. Boston, MA: Allyn & Bacon.

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

Bevan, J., Werner, L., McDowell, C. (2002). Guidelines for the use of pair programming in a freshman programming class, Fifteenth Conference on Software Engineering Education and Training (100-107). Covington, Kentucky, USA: IEEE Computer Society Press.

Cockburn, A. & Williams, L. (2002). The costs and benefits of pair programming" in Marchesi, M. and Succi, G. (Eds.), Extreme programming examined, Boston: Addison-Wesley.

Collings, P. & Walker, D. (1995). Equity issues in computer-based collaboration: Looking beyond surface indicators. In Proceedings of CSCL '95: The First International Conference on Computer Collaborative Learning, Schnase, J.L. , Cunnius, E.L., eds. Mahwah: Lawrence Erlbaum Associates, Inc., pp. 75-83.

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

Fennema, E., & Sherman, J.A. (1976). 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, 6(31), (Ms. No. 1225).

Jehng, J. c. J. (1997). The psycho-social processes and cognitive effects of peer-based collaborative interactions with computers. Journal of Educational Computing Research, 17, 19-46.

Lips, H. M., & Temple, L. (1990). Majoring in computer science: causal models for women and men. Research in Higher Education, 31(1), 99-113.

MacGregor, S. K. (1988). Computer programming instruction: effects of collaboration and structured design mileposts. Journal of Research on Computing in Education, 21, 155-64.

McDowell, C., Werner, L., Bullock, H., Fernald, J. (2002). The effects of pair programming on performance in an introductory programming course, Proceedings of the Conference of the Special Interest Group of Computer Science Educators (pp. 38-42), Northern Kentucy: ACM Press.

Oliver, R., & Malone, J. (1993). The influence of instruction and activity on the development of semantic programming knowledge. Journal of Research on Computing in Education, 25, 521-33.

Priebe, R. (1997). The effects of cooperative learning in a second-semester. Paper presented at the Annual Meeting of the National Association for Research in Science Teaching, Chicago, IL.

Romeu, J. L., & Alemzadeh, J. (1998). A statistical assessment of an experiment to compare traditional vs. laboratory approach in teaching introductory computer programming. Journal of Educational Technology Systems, 27, 319-24.

Roth, W.-M. (1993). Construction sites: Science labs and classrooms. In K. Tobin (Ed.), The practice of constructivism in science education, (pp. 145-170). Hillsdale, NJ: Erlbaum.

Slavin, R. (1980). Using student team learning. Baltimore: The Center for Social Organization of Schools, The Johns Hopkins University.

Slavin, R. (1990). Cooperative learning: theory, research, and practice. New Jersey: Prentice Hall.

Smith, K. B. (1994). Studying different methods of technology integration for teaching problem solving with systems of equations and inequalities and linear Programming. Journal of Computers in Mathematics and Science Teaching, 13(4), 465-79.

Webb, N.M., Nemer, K.M., Chizhik, A.W. (1998). Equity issues in collaborative group assessment: Group composition and performance. American Educational Research Journal, 35(4), 607-651.

Williams, L., Kessler, R., Cunningham, W., & Jeffries, R. (2000, July/August 2000). Strengthening the case for pair programming. IEEE Software, 17(3).

Williams, L. A. (2000). The Collaborative Software Process PhD Dissertation. Doctoral Dissertation, University of Utah, Salt Lake City, UT.

Williams, L. A., & Kessler, R. R. (2000). The effects of "pair-pressure" and "pair-learning" on software engineering education. Thirteenth Conference on Software Engineering Education and Training (pp. 59-65), Austin, TX: IEEE Computer Society Press.

Williams, L. A., & Kessler, R. R. (2001, March). Experimenting with industry's "pair programming" model in the computer science classroom. Computer Science Education, (1) 7-20.

Yusof, Y. B. T. M., & Tall, D. (1998). Changing Attitudes to University Mathematics through Problem Solving. Educational Studies in Mathematics, 37(1), 67-82.