Pair Programming: More Learning and Less Anxiety in a First Programming Course

Susan F. Freeman, Ph.D. *, Beverly K. Jaeger, Ph.D.*, Jennifer C. Brougham[✦]
***Northeastern University, [✦]Monash University**

Pair Programming is a recent development in education designed to enhance the student's learning experience through teamwork. Specifically, it involves the students undertaking and completing some aspect of their programming work as a team of two. In engineering it is generally utilized for computer coding projects, such that partners work conjointly on the same design, algorithm, code, or assignment. For this study, this approach was used in a first programming course entitled "Engineering Problem Solving with Computation" by adapting the XP: eXtreme Programming method that is used in industry. The objective of this course is to teach first-year engineering students logic and problem solving skills through algorithmic programming. The programming language in the course of interest was C++. Pair programming was a voluntary element of the course and the students, under certain conditions, could opt in or out of it. The primary objective in offering this option was to reduce students' frustration when programming for the first time, as this often causes them to abandon attempts to understand the nuances of programming logic and syntax, and thereby the intended educational value of the course is lost. Pair-programming methodology requires that participants assume specified roles in a partnership. The course is administered and this option is selected with this understanding. Through various assessment tools, it was found that pair programming inspired confidence in the students, specifically in their ability to achieve the task at hand. Students reported that they benefited from being exposed to their partner's ideas and suggestions, and that they therefore broadened their understanding of the assignments' requirements. The students indicated that it was easier and quicker to complete their work and there was an overwhelming belief expressed that it helped them identify errors more readily and consider alternative approaches to problem solving. Objective analyses taking into consideration the students' grades and timesheets support, at a *prima facie* level, the students' accounts of their experiences. From a pedagogical perspective, pair programming offered better use of the university's facilities, whereby fewer workstations were required, and the instructors' time was better utilized, reducing the number of potential inquiries and improving the quality of inquiry to be addressed. Foremost, the students themselves also reported a positive experience of being able to solve minor problems, such as syntax errors, by the immediate presence of their partner, which enabled higher levels of learning to be achieved. In addition to the qualitative and quantitative advantages of pair programming, this paper will discuss challenges and difficulties that surfaced and will outline the projected future of this methodology for engineering education.

INTRODUCTION

A large percentage of engineering success is built on teamwork. The benefits of collaboration are particularly evident when developing computer code to solve complex problems, provided that the programmers possess a base level of engineering logic, computer language literacy, and communication skills. With this in mind, are there potential benefits to be derived from teamwork for novice programmers? Is it possible and is it easier to teach computer-based problem solving to new engineers by having them work in pairs?

Recent literature supports the notion that student programmers who participate in team learning and practice *pair programming* tend to perform better on coding projects and are more likely to succeed in early code development than if they had worked alone (Williams *et al*., 2001). Based on multiple reports from research in this methodology, it was anticipated that an adapted pair-programming experience, if properly implemented and observed, would improve logical thinking, code quality, and the confidence of working partners in a first-year engineering programming course. Other potential benefits in both professional and academic settings are outlined below and evaluated in this investigation.

BACKGROUND

Advantages of structured collaborative programming are well established in the literature and in its application in industry. This technique, known as the eXtreme Programming (XP) method, (Beck, 2000) was initiated for software development in a corporate setting. XP promotes a systematic and shared approach to generating computer code. With XP, programmers follow a code development structure and take on predefined responsibilities to generate code as a team. Pair programming adapts the XP approach to attain educational results and enhance the learning process primarily through role assignment.

Some established benefits of XP and its adaptations, such as pair programming, include an increase in coding error detection, a reduction in final program coding defects and a reported increase in collaborator learning and satisfaction. Participants in pair programming also note improved morale with enhanced team-building and problem-solving skills (Williams & Upchurch, 2001). Some corporate advocates of the structured XP methodology are so convinced of the advantages of structured and paired code development that they have declared that all production code must be written in partnerships (Wikki, 1999).

The Roles: Driver and Navigator

Pair programming is not merely an exercise in dividing up required work on a coding project. For each program, a member of the pair has a specifically defined role of being either driver or navigator as set forth by Williams and Kessler (2000). In the definition of roles, students are expected to adhere to stipulated operative guidelines: The driver has control of the pencil, mouse, or keyboard and writes the code. The navigator continuously and actively monitors the work of the driver, watching for defects, thinking of alternatives, consulting resources, and considering strategic implications of the work. The team members are not to break the assignment into parts and integrate it later. Partners are expected to schedule time together for the purpose of completing programming work.

Williams (2000) surveyed professional programmers on this practice, finding that 100% agreed that they had more confidence in their solutions when engaging in pair programming than when programming alone and 96% agreed that they enjoy their job more when programming in pairs. As reported by Beck (2000) and Wikki (1999), many attribute their professional coding success to the teamwork approach.

Pair programming has been applied and tested in the university classrooms as well. Two upper level programming courses at the University of Utah utilized the collaborative approach in web and software development courses. In both courses it was found that paired students performed much more consistently and produced higher quality work than they did individually. Two compelling responses emerged from student surveys as motivating factors. They were: "between us we could figure most things out" (74%) and "there was pair-pressure to not let my partner down" (63%). An overwhelming 95% of students felt that they were more confident in their assignments because they pair programmed and in general felt they were more productive in the collaborative setting. A large majority of students agreed that they remained more focused and enjoyed the assignments more because of pair programming.

In the University of Utah work, pair learners passed significantly more post development test cases than those who performed and learned individually. The Utah study reported that the total hours invested working on programs was, in effect, the same across each of the groups.

Similar research was done at North Carolina (NC) State University to evaluate the efficacy of pair programming in an introductory computer science course (Williams *et al*., 2001). Of note was the higher success rate in the course and significantly better performance on 2 out of 3 programming projects by pairs. They further reported that there were no significant differences between the paired and solo course sections on midterm and final exam scores. This indicates that pair programming did not detract from the students' educational experience.

The NC State study also evaluated perceived course and instructor effectiveness, finding that the students who engaged in pair programming rated both the course and the instructor as more effective. Surveys administered to measure self confidence, motivation, and attitudes toward course elements surprisingly revealed no differences between groups exposed to paired versus solo learning modes.

Novice undergraduate programmers at the University of California Santa Cruz demonstrated better performance on programming assignments when working in pairs. Also, improved retention rates have been attributed to team programming techniques (McDowell *et al*., 2002). They found that an appreciably higher percentage of students in the partnered class took the final exam as compared with the solo students. Yet, they found no significant difference between groups on exam scores.

Objective Outcomes

As noted, quantifiable outcomes that could arise from pair programming which are potentially beneficial to the team are: (1) earn better grades on programs and in the course overall, with a possible carry-over effect to quizzes and exams, depending upon their programming content; (2) demonstrate more efficient completion time for programs; (3) utilize fewer workstations, less computing and possibly less printing resources; (4) enhance the teacher/team and teacher/program ratios in the instructional environment.

Subjective Effects

The benefits described above were anticipated in addition to those more specific to first-year engineering students, including that pair programming would: (1) decrease anxieties of writing first program for novice programmers, (2) minimize high frustration levels often observed with novice programming activities, (3) reduce non-approved collaboration behavior and academic dishonesty temptations and issues, and (4) provide a more supportive and less competitive environment in terms of academic achievement.

Initial experimentation with pair learning has produced benefits for instructors as well (Williams & Kessler, 2002). Students working in pairs are able to answer one another's questions, since the faculty is no longer their only source of technical guidance. The classroom tone is calmer because students are more self-sufficient and, as noted above, the teacher/team ratio is lower. Also, the grading load is substantially reduced as pairs submit one assignment.

Applied Effects

Teamwork is advocated and encouraged in many engineering educational settings, which include project work, tutoring, study groups and co-op assignments (work in industry outside the school setting). Since collaboration is encouraged in engineering industry practices and programming projects in the corporate sector, this approach compliments the students' desire to prepare for subsequent entrance into their careers.

METHODOLOGY

Participants

A total of 128 students enrolled in five sections of an introductory engineering course, Engineering Problem Solving with Computation, participated in the study. This is a required course taken in the third quarter of the first year engineering program at Northeastern University. This module of the curriculum possesses a dual emphasis on learning a programming language and solving and analyzing real-world problems through the formulation of computer code.

Procedures

The pair programming and solo options were described to all students verbally and on a document that was distributed early in the course. Part of an initial homework assignment was to develop a list of concerns and questions about teamwork versus solo alternatives. Each of these issues was addressed across the course sections to ensure that all students obtained the same introductory information about the available options.

The first coding assignment was a programming tutorial in which all students worked independently by design. The second assignment required the students to engage in pair programming roles to develop two basic computer programs. The pairs were to trade driver/ navigator roles after the first program and keep time logs on a table provided by the professors. After the second programming assignment, students could opt to work with a classmate of their preference under the pair-programming guidelines or decide to remain solo.

A survey was administered after 3 assignments to determine if any student requested a change of status or partner, although participants were expected to work within their selected options and keep time logs for a total of 6 programming assignments. The students were made aware of this option in the introductory sessions. All students elected to retain their pair or solo status; some indicated a willingness to change partners, but there were no formal requests for new partners at the midpoint. Consequentially, no shifts were made other than two students who discontinued in the course. Their data is not included in this study.

Students worked on weekly assignments, either solo or in their teams. At least one class period of 3 hours per week was allocated for students to work on their assignments. Student teams were requested to arrange weekly meeting times prior to leaving the computer lab and to honor them outside of class. Primary requirements for the two options were: (1) to adhere to driver and navigator roles as pairs or refrain from unauthorized collaboration as solos, and (2) keep accurate records of time invested in each portion of the assignment. Pairs also confidentially reported the perceived contribution by their partner to each assignment. At the instructor's discretion, a student's participation in pair programming was contingent upon acceptable progress, contribution and performance in the course.

All students took the weekly quizzes and the final exam individually. Only the assigned homework programs were completed under the elective status conditions of paired or solo. All students completed time logs, which reflected the time spent on each programming project in and out of class. Teams responded to questionnaires and surveys about their experiences.

Data Collected

Periodic qualitative and quantitative assessment tools were used to evaluate the students' progress and satisfaction, as well as the program's effectiveness. Four types of data were collected during the course of this investigation: (1) average grade values, (2) time and contribution records, (3) results from preference and opinion surveys, and (4) responses from open-ended questionnaires.

Pairs versus solos were compared on the objective metrics. Comparisons were also made using cohort data from previous classes taught by one of the author instructors on the extent of topics covered and grades achieved. Faculty and student commentary from traditional sections (all solo) were also reviewed.

RESULTS & DISCUSSION

The division of students across the learning modes is presented in Table 1. The numbers translate to a 41% reduction in computer workstation requirements and concomitant computer processing resources. This also resulted in an appreciable reduction in the number of inquiries made to the instructor and projects to grade.

Table 1. Paired vs. Solo Selections Across All Sections

Total # of Students	Total Pairs	Total Solos	Required # of Work-stations	% Reduction in Resources
128	53	22	75	41%

Material Covered - Learned Topics

The instructors conducting this course are members of a cohesive teaching team so as to ensure that the students across all sections of the course are exposed to the same foundation of subject matter. Each concurrent section covered material comparable to previous years and comparable to one another in extent and detail. As such, all predetermined topics were presented and a common final exam was administered across all sections.

Grades

Table 2 show students' average grades for homework, weekly quizzes, the final exam and the overall course average. In each grade category, a *t*-test was performed to compare the pairs and solos for the current course sections. The *t*-test results show that each group's performance on the quizzes, final exam and overall course grade were not significantly different.

Table 2. Grade Comparisons Across Groups

GROUP AVERAGES (out of 100)	Home-work Average	Weekly Quiz Average	Final Exam Average	Overall Course Average
Average Pairs	91.26	79.58	79.34	85.05
Average Solo	79.68	80.22	78.11	80.27
Average Previous Year	85.60	80.37	82.14	84.18
t-test: Pair vs. Solo	*.056*	*.89*	*.80*	*.29*

The mean achievement for pair programmers on coding homework is visibly better than that of the solo programming group, and the difference is nearly significant at p=.056. These outcomes are similar to the results of McDowell *et al.* (2002) and Williams *et al.* (2001), who did not find significant differences in work quality between pairs and solos.

The instructors note that the majority of solo programmers were more experienced in programming, and/or comprised a small percentage who were not motivated and therefore not desirable project partners. That may explain why the solo quiz average is slightly higher, but their homework averages are lower. Such students tended to be apathetic about homework, yet were sufficiently knowledgeable to achieve well on the quizzes. The first year student's course load is quite heavy, so they often make a decision not to do homework that they perceive as tedious. Comparing last year's averages to the current year in the four grade categories, the values lie between the pair and solo averages except for performance on the final exam. Variations in the exam itself could account for the differences on this metric.

Grades are only one indication of what the students have learned. The goal was for the students to learn at least as much as they would have learned without pair programming, but with the added bonus of less frustration for all involved. As discussed previously, the amount learned in the course was at least equivalent to that learned by the cohort sections. The higher level of performance on homework indicates that the students are practicing programming more, and giving up less. Instructors have reported in previous years that the students would encounter difficult areas and abandon projects out of frustration, whereas, as noted in sections of this paper and in other studies as referred to above, under the pair-programming paradigm, the paired students did not want to let their partner down and therefore completed their work. Further, because of the teamwork, pairs were able to get past roadblocks more easily that would normally require assistance from teaching staff, using their own combined knowledge as tools. This increased success resulted in more homework being completed, more being turned in, in better form, and consequentially earning higher scores.

Time Invested

All students, regardless of status, were required to maintain and submit time logs. Participants recorded the date, location, hours worked, student's name(s) and on which component of the assignment they worked. Table 3 shows that there is no strong pattern for the difference in time spent on the assignment between solo and pair programmers, with no significant time differences found between the learning modes at the p<.05 level of significance.

Table 3. Average Time Spent on a Programming Project

Section Number	Average* for class	Average* for pairs	Average* for solo
1‡	6.2	6.3	6.1
2‡	4.2	3.7	5.0
3	4.4	4.4	N/A
4‡	4.4	4.2	5.1
5‡	4.2	4.3	4.1

*All values were in hours, recorded to nearest 15-minute increment
‡ No significant differences were found, a=0.05

In one class, solo programmers took longer, and in another, the pair programmers took longer. The differences are small, indicating that pair programming does not result in an overall reduced time investment by the team members. As evidenced by their feedback below, the majority of students perceive that pair programming is procedurally more efficient, but there is no quantitative evidence to support this notion. There are two confounding factors to consider here: (1) since the solos tended to reportedly excel at programming logic and/or had significant prior coding experience and preferred to work alone, the time invested may be an artifact of the elective solo status, and (2) it may be that each member of a pair, if working solo, may have individually taken longer, but this cannot be measured.

It is important to note that pair programming did not result in an overall increased time investment by the team members. Initial concerns of the students that discussion by the pair members will add undue additional time to their workload can be countered by this data. Reduction and detection of errors and concurrent research facilitated by the navigator helps in this respect. To this extent, as with the students' grades, we can say that pair programming does not detract from the students' educational experience on a time basis, and therefore its other benefits can still be enjoyed.

Given that more homework was submitted under pair programming, it is likely that it was completed earlier and with a decreased need for instructor intervention. Since fewer students needed to wait to have questions answered outside of class, they could proceed and finish sooner on a more efficient basis rather than utilizing the same time period spread across several days while waiting to obtain answers from faculty. This advantage is not measured in this study, but has been reported anecdotally by the students and may also explain the difference between the actual time and efficiency perception of the students.

Partner Contribution

On a number of assignments, pair programmers were asked to confidentially report the percentage of work each partner had contributed. Each team member was asked to respond separately and independently. This was done to assess whether the structured roles and rules of pair programming were being followed and to address any imbalances or scheduling problems with partnered teams. All of these responses were combined and tallied and are presented in Figure 1.

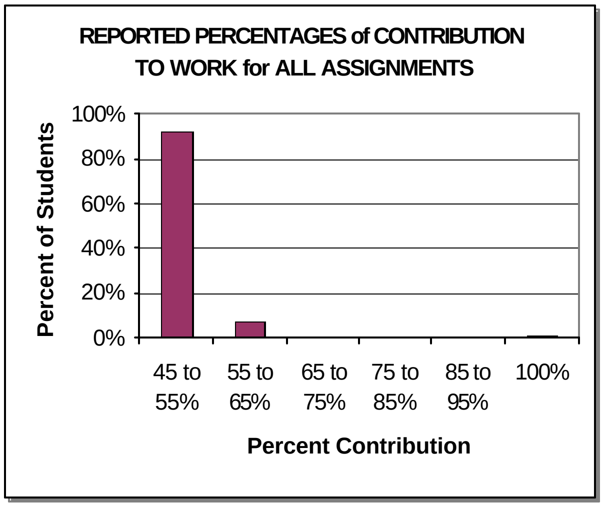

Figure 1. Reported percentage of contribution to work.

Over all of the contribution responses, 91% reported that the workload was distributed evenly (50/50 $\pm$.05). A small percent (7%) felt that the distribution of the workload was closer to 60/40. There were a few 100/0 occurrences (<1%). In these rare instances, the team and its status were re-evaluated by the instructor to resolve the imbalance. In general, students were sharing the workload evenly, thereby achieving one of the goals identified and sought at the outset, and confirming a proposed benefit of pair programming.

Student Surveys

Students who voluntarily engaged in pair programming responded to a series of survey questions. From content analysis of responses to the open inquiry, "What did you like about pair programming?", distinct categories emerged. As seen in Figure 2, the majority of remarks revolved around three clusters of response types: effective learning, teamwork, and time efficiency.

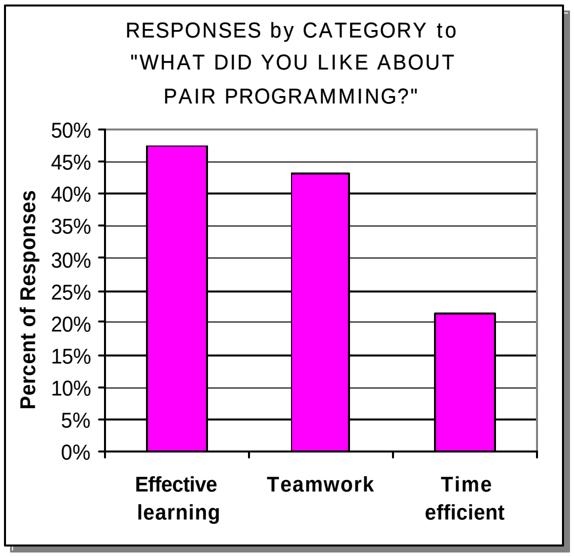

Figure 2. Response clusters to open inquiry, "What did you like about pair programming?" at course conclusion.

The categories seen in Figure 2 emphasize some of the important benefits of pair programming, which the participants identified without instructor prompting. In the first cluster, 47% stated that they felt this was a more effective method of learning for them; while almost as many, 43%, enjoyed the teamwork and collaborative dynamics of pair programming. Over 20% believed that they were more efficient due to the team approach. A large number used the word "enjoyed" in connection with their feelings for a programming class which, in one instructor’s experience, has never happened before. Since students could make numerous comments spanning multiple categories, the total is over 100%.

At the end of the course, several sections of students were given the list of comments shown in Table 4 and could check off as many of them as they felt applied. Admittedly, it was easy to check them all off, but students were given time to consider each one. The results show resoundingly that the participants felt very positively about pair programming.

Over 90% felt that they had done better quality work, were more confident about their work, and would recommend that it be continued. Nearly 90% praised the efficiency element and felt that they had learned more. Combined with the other comments that were reviewed, and the large volume of written material where the students praised the method, this chart reinforces the inference that students had a highly positive experience using pair programming.

Table 4: Checklist Survey Responses

Response Selections for Pair-Programming Students	Percent
Our programs were of better quality because we pair programmed.	93.88%
I was more confident in our assignments because we pair programmed.	91.87%
I would recommend pair programming be continued in future classes.	91.84%
We were more productive or efficient because of pair programming.	89.80%
I learned more from explaining my work to my partner.	85.71%
I learned more because my partner explained their work to me.	85.71%
I enjoyed doing the assignments more because of pair programming.	75.51%

Use of Resources: Anecdotal Information

The calculated 41% decrease in computer demand was accompanied by an equivalent reduction in the number of programs that an instructor was required to oversee. Fewer questions were posed to the instructors, who also reported that the nature of the inquiries was more advanced and of higher quality than those encountered in previous solo courses. Further, the individuals who elected to work independently enjoyed the benefit of the professors' enhanced availability, which improved their learning potential as well.

Many of the open responses to what students liked about pair programming were so similar as to be considered identical and were collected for Figure 3. All of the positive comments could not be recorded here, but this chart shows that the majority of the feedback was very encouraging, when considering the effect on their learning, their time, and the quality of the experience.

There was a large volume of written material from the student surveys and in memorandums written about the course to their instructors. A sampling of these quotes summarizes the positive feedback and general attitude of the majority of the students:

- *When we work in pairs, there are two points of view which can provide richness and solve problems easily.*
- *Without the roles that we were working in, finishing this assignment would have been much more difficult. Instead of working with one person's ideas, we used two sets of ideas, and pooled them together in order to produce an answer to the assignment.*
- *It helps your communication between fellow students, and improves your ability to work with others.*
- *Not only are the assignments less stressful and more time efficient, but also it is a better way to learn material because the [partner] acts as another resource*
- *The pair program seems to be a success because it is easier to collaborate between two people instead of just asking for help on our work from one another.*
- *Through pooled efforts we taught each other sections of the material that may have been unclear to us.*
- *We were less frustrated than working alone. Many times when one person has been working on something, he cannot tell what he did wrong. As soon as another person steps in, he may see what is wrong.*
- *The pair programming is a great way to learn teamwork and get to know the other students.*

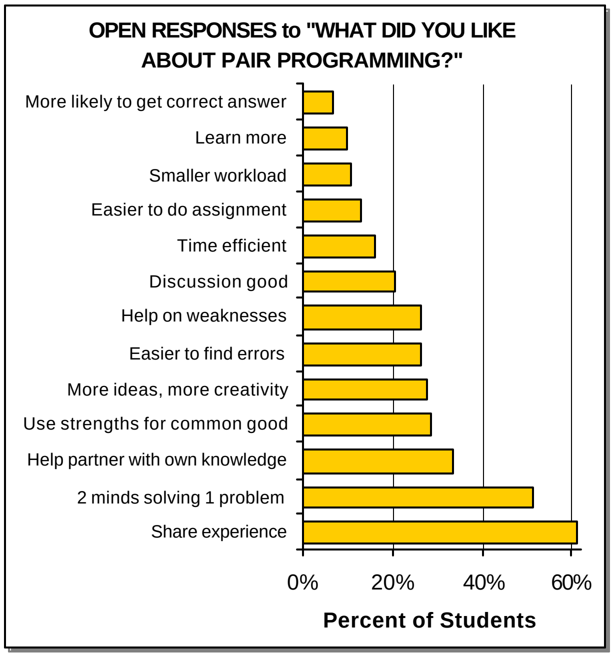

Figure 3. Open responses from students advocating pair programming methods.

Class Demeanor: Anecdotal Information

The instructors, experienced in teaching programming, noted that the tone of the lab environment was distinctly more collegial and less stressful than in past courses. Students conveyed a higher perceived level of satisfaction in the course and with the projects under the elective pairing paradigm. As evidenced by the homework averages, students reportedly gave up less and were more likely to submit more complete projects than when working alone. Not surprisingly, there were no instances of confirmed or suspected academic dishonesty. All of these outcomes were personally and professionally rewarding for the instructors.

Concerns and Challenges

Implementing a collaborative learning methodology as described in this work does not occur in the absence of obstacles and drawbacks. Some of the concerns that were expressed by the students in their initial inquiries and recognized by the instructors at the outset were unavoidable. Those issues that were not entirely resolvable are listed in Figure 4 along with further recommendations from the paired participants.

In addition to asking the students what they liked about pair programming, they were also asked, "What would you change about the pair-programming system?" Over 75% of the paired students answered "nothing" or made no comment and such responses were combined into one joint category in the data results. These responses were combined for 2 reasons: first, as these students were asked for a large volume of feedback and they generally provided such details when a problem was perceived, 'no response' was taken as indicating no problem existed, and second, as first year students, this category of students are generally inclined to respond if and when there is a problem and no additional response is volunteered. Accordingly, in most if not all cases, "no comment" does equate with the reply of "nothing".

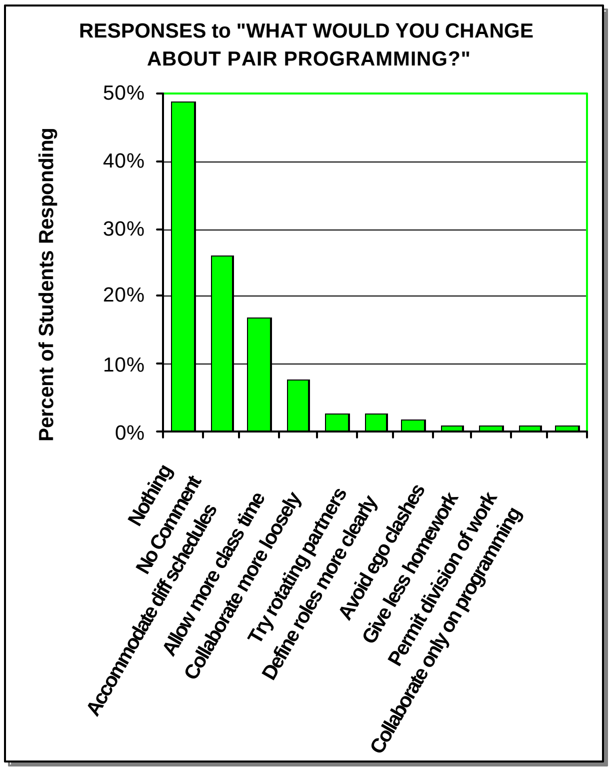

Figure 4. Recommendations from students for modifications to pair programming implementation.

A variety of recommendations were proposed by the participants. There were concerns with partners having scheduling difficulties and finding a mutual time to meet. Some responses revolved around role problems with their partners and the requirements of the driver and navigator. Other suggestions concerned the division of tasks; it was generally agreed that collaboration should be reserved for the programming component of the projects and that it should not be compulsory to work jointly on the remainder of the work elements, such as letters and summaries. Even with these comments, the majority of students did not identify any serious issues with programming in pairs.

As noted, a few comments reveal that some partners were experiencing scheduling difficulties given their busy schedules:

- *It's easier with my schedule to do things when I can – meeting times have been hard to do and I feel like I could finish the programs faster on my own.*
- *I couldn't find a lot of time to meet with my partner because I am commuting.*
- *We had difficulties in trying to arrange a time to meet and a place to work.*
- *On the other hand arranging a time for working as a pair is not easy since everybody has his own schedule. Also there can be problems about meeting on time…*

From the instructor's perspective it is impossible to monitor pair behavior, "role-keeping", and scheduling outside of class, regardless of how well defined the responsibilities may be. Likewise, some of the assignments were less cohesive when students divided up administrative tasks, resulting in a more fragmented distribution of labor. This was partially managed by dedicating class time and scheduling supervised lab sessions during the students' free time and by reviewing time logs and peer evaluations.

Most pairs were able to overcome the scheduling challenge, but some decided to divert from the directive that they must always work together in defined roles. The following comments regarding problems and areas of change in pair programming illustrate this other area of concern:

- *Maybe have the roles divided a little more clearly.*
- *I would not make it mandatory for the pair to work together on every task.*
- *To attempt to evenly divide the workload, we each chose the assignments that we were more comfortable with. This made it easier to finish the assignment, and it was accomplished in a more timely manner..*
- *Navigator interfering by asking too many questions about your logic.*

The comments indicated that at times the pairs split the work into tasks, which they then did separately. As a result, some of the assignments were not as cohesive when students divided up the administrative tasks.

Several pairs mentioned that writing memorandums and letters together was difficult and would like to relax that constraint. A few of the statements noted conflicts and difficulties merging ideas into one program. Sometimes the concern was dividing the work, and finding a switching point. A few students reported that they were not always clear on the roles of driver and navigator, and it seems some may not have switched roles as often as directed. It is important to note that the comments identifying problems were rare in comparison to the number of positive responses received throughout the study. These comments also remind the instructor of the importance of initially defining the roles and then regularly reinforcing those roles to the students to ensure compliance.

Another area of concern is how to accurately and objectively evaluate a student's competency and progress with the course content. As described, weekly quizzes and the final course exam were administered independently and had significant contribution to each student's final course average, along with daily assignments and attendance. Weekly programming projects accounted for 35% of the student's course grade. It was decided that any large discrepancies in individual grades or attendance problems would be addressed immediately by the professor and could jeopardize a student's status in the program.

Recommendations

Upon reflection on the feedback from the students and the instructors' experiences, we recommend providing the students with a detailed document to read about the course, noting its benefits and how the course is to be conducted, to reduce some of the concerns with students adhering to the specified roles in pair programming. The document should include specific guidelines with explanatory samples of the documents to be completed by the students, and include testimony and commentary from past participants. The written information should be followed by a session allowing students to ask questions. These procedures can be complimented with reminders throughout the course regarding the importance of time logs, and how and when to divide work and swap roles.

It would also be helpful to provide more time in class under supervision to have students engage in pair-programming activities. Open lab sessions in which the instructor was available to help were piloted and were well utilized by the students, and appeared to increase the percentage of time they worked effectively as pairs. This could be implemented more rigorously in future endeavors, with regular reminders to the students of its availability.

To better understand whether there are time benefits from pair programming, a counterbalanced course could be taught where each student is required to do both solo and paired work, so a time comparison can be made on the specific students themselves, rather than across different groups with their own different internal skill levels. The students may be reluctant to do unpaired work, but this would provide better data for evaluating the time effectiveness of pair programming.

Further research efforts to support and expand the implementation of pair programming should include a longitudinal study to evaluate the long-term effectiveness of the method in subsequent academic or professional application. This should be accompanied by continued assessment of the existing methods, implementation of necessary adaptations, and review of the utility of any modifications.

CONCLUSIONS

The objectives of pair programming are to benefit the students by enhancing the quality and quantity of their learning and also allow the Engineering instructors to be more effective in their educational roles.

Benefits for Students

One of the principal objectives in developing the course to incorporate pair programming was that it was hoped that it would provide "improved success on programs and better performance on tests, [and] decreased frustration". Whilst there are method problems in specifically evaluating the grades of the students, it is fair to say that the paired teams certainly performed as well as the solo programmers and, in some instances, the partnered students were able to be more successful because of their work done in pairs. It gave them the opportunity and capacity to achieve under more supportive circumstances.

The remarks from the students themselves provide us with a better view as to how pair programming was working for them. There were frequent comments that the students felt that it was less stressful for them and they agreed that there was decreased frustration. Further, participants noted that it helped them to learn more by being able to identify the errors more readily and being able to solve the problems with the assistance of a partner. They enjoyed being exposed to an alternative perspective in problem solving and often expressed a viewpoint that the course was enjoyable and successful for them.

The prime value of the option to pair program lies in its ability to help the students focus on the engineering objectives of the course and not be burdened and effectively overwhelmed by minor typographical and syntax issues. In this sense, the role of the navigator was particularly valuable. It was this aspect that helped the students minimize their error rate and provided them with the opportunity to view the work being completed from a different perspective whilst still being involved with the project. The ability to stand back and "see the forest" without having to deal directly with each individual "tree" seemed to be a positive and constructive educational experience not consistently experienced by solo novice programmers.

Benefits to the Engineering Department

The goals of this project also included improving conditions for the faculty, so that they can be better utilized in the lab and improving the teacher/program ratio. Accordingly, it is possible to have increased class sizes without detrimentally affecting the students' learning. A direct economic benefit from pair programming is the reduced number of computer stations and licenses that would be required for the course. These numbers also have an immediate effect on the faculty, in that it would be unlikely that all of the paired students would seek individual help on a constant basis; it would be more likely that they would consult the faculty on assignment issues in pairs, therefore the number of inquiries and assistance sought from the faculty would be and was reduced substantially.

In addition to the resource reduction, the students in their own commentary noted that they had to seek the assistance of the instructors less because of the combined knowledge of the team. Therefore, the larger information base provided by each pair reduced the number of problems that remained unresolved. Likewise, their increased capacity to detect errors resulted in fewer issues created by minor syntax, typographical, or grammatical mistakes. All of these factors combined meant that the faculty were left with more time to address issues of substance and properly address them without the pressure of leaving other students stranded waiting for their attention. In this sense, it provided a potentially higher quality academic experience for all students, paired or solo.

With pair programming, leveraging the power of two minds in this role-driven team approach can make computer coding more accessible to those with minimal or no background in this field. In the educational realm, it also has the potential to enhance the collaborative work experience for all levels of programming aptitude, contributing to program quality and programmer confidence.

REFERENCES

Beck, K. (2000*). Extreme Programming Explained: Embrace Change*. Reading, MA: Addison-Wesley.

Bennis, W., Biederman, P. (1998). *Organizing Genius: The Secrets of Creative Collaboration*. Cambridge, MA: Perseus Publishing.

Nawrocki, J. & Wojciechowski, A. (2001). *Technical Paper, KBN* Grant 8T11AO1618.

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 (SIGCSE).*

Nosek, J.T. (1998). The Case for Collaborative Programming. *Communications of the ACM*, 41:3, 105-108.0

Wikki, (1999). Pair Programming. *Portland Pattern Repository, June.* http://c2.com/cgi/wiki/ProgrammingInPairs.

Williams, L.A. (2000). The Collaborative Software Process. *Ph.D. Dissertation*. University of Utah, Salt Lake City.

Williams, L.A., Kessler, R.R., & Cunningham, W, & Jeffries, R. (2000). Strengthening the Case for Pair Programming. *IEEE Software 17:4,* 19-25.

Williams, L.A. & Kessler, R.R. (2001). The Effects of "Pair-Pressure" and "Pair-Learning" on Software Engineering Education. Conference on Software Engineering Education and Training 2000.

Williams, L.A. & Kessler, R.R. (2000) Experimenting with Industry's "Pair-Programming" model in the computer science classroom. *Journal of Computer Science Education, December.*

Williams, L.A. & Upchurch, R.L. (2001). In Support of Student Pair Programming. 2001 SIGCSE Conference on Computer Science Education, Charlotte, NC, February 2001.

Williams, L.A, Wiebe, E. Yang, K., Ferzli, Miller, C. (2001). In support of Pair Programming in the Introductory Computer Science Course. Computer Science Education, September, 2002.

Williams, L. Pair Programming Questionnaire, 1999. Available at http:\\www.pairprogramming.com.

http://www.cs.put.pl/awojciechowski/research/pair_programming/ Reviewed January 2003.