Experimenting with Pair Programming in the Classroom

Charlie McDowell, Brian Hanks, and Linda Werner
Computer Science Department, University of California
Santa Cruz, CA 95064
{charlie,brianh,linda}@cs.ucsc.edu

ABSTRACT

There is now a substantial body of evidence in support of the use of pair programming in the classroom[3, 4, 10, 11, 13, 14]. Some of the data is anecdotal and some is the result of formal experiments. We are not aware of any published data that raises concerns about allowing students to complete programming projects using pair programming.

In this paper we present data from three studies performed at UCSC. All three studies support the position that pair programming results in more student learning.

Categories and Subject Descriptors

D.2.0 [**Software Engineering**]: General. K.5.1. [**Computers and Education**]: Computer and Information Science Education–*computer science education.*

General Terms

Experimentation, Human Factors

Keywords

Pair Programming, Extreme Programming

1. INTRODUCTION

Essentially all non-trivial software projects are collaborative efforts. Most professional programmers, at least on occasion, look over the shoulder of a colleague to help them solve a programming problem. In recent years this informal and occasional sharing of a workstation has become more formalized and widespread, largely due to the development of extreme programming (XP)[1]. All software developed using XP is worked on collaboratively by a pair of developers. While programming, the pair work side by side at a single workstation with one person designated as the 'driver' and the other person as the 'observer' or 'navigator'. The driver has control over the keyboard and mouse and is responsible for entering program code. The observer role is not passive; observers watch for potential defects and comment about programming approaches. These roles are switched as the programming session continues.

Unlike software developed by professionals, most programs written by college students are written by a single programmer, as required by the instructor. The commonly held belief has been that the students must write the programs on their own in order to learn how to program. This belief includes the assumption that if allowed to work with a partner one student might do all of the work and the learning while the other student does neither.

Some classes, typically the more advanced classes, involve group projects. But in most of these group projects, the collaboration is limited to design and specification. The coding is generally done by individual students and then integrated near the end of the project.

This solitary programming approach has begun to change in recent years with growing numbers of instructors requiring or allowing students to use pair programming. Pair programming is very different from a two-person team project. Historically, on a team project the students would be encouraged to use divide and conquer (e.g., you write the scanner and I'll write the parser). With pair programming all code is developed at a single workstation with both students working together.

ITiCSE'03, June 30-July 2, 2003, Thessaloniki, Greece.

2. PREVIOUS STUDIES

Recent findings now question the long followed practice of requiring students to complete programming projects individually[3, 11, 14]. Although some of the findings are anecdotal, others are the result of formal experimentation. The published studies on pair programming in the classroom have identified a number of reasons why instructors should allow their students to use pair programming. These benefits include more students passing the course, higher quality programs, less time to complete programming projects, increased student satisfaction, increased numbers of students continuing with a computer related major, and possibly better exam scores (when drop rates are factored in). The amount of benefit has varied due to different classroom environments. The classes have ranged from introductory programming classes[3, 11] to senior software engineering classes[7, 14]. Another important variable is the type of laboratory environment. In one experiment a significant portion of the programming was done in a controlled or closed lab with careful supervision of the pair programming process[14]. In another the students received little or no direct supervision of the pair programming process[2].

The nature of the experiments has also varied. In the ideal scenario, students would be randomly assigned to work in pairs or work alone. The students would otherwise receive the same instruction and the instructor would not know who was pairing and who was not. Such a scenario is impractical. In practice the experiments have covered a range of methods. In one experiment all students in the class were assigned to one of two groups (pairing or individual)[14]. In other experiments all students in one class were required to use pair programming and in another offering of the same course students were not allowed to use pair programming[3, 11]. In some of the classes described in this paper students were simply allowed to use pair programming. We also report on a small study in which volunteers were randomly assigned to a pairing group and a non-pairing group. Needless to

say, the types of conclusions that can be drawn differ for these different methods. What the experiments all have in common is that they all conclude that pair programming benefits students.

3. THREE STUDIES AT UCSC

We have conducted a series of experiments on the use of pair programming by students. Our studies differ from some other studies in that our students received little or no supervision of their pair programming. They did receive a brief explanation of what pair programming is, and they were instructed to read a paper on pair programming[12]. We have performed three related but slightly different studies.

3.1. Study 1: Voluntary Pairing

Data for the first study was collected in winter 2001 on an advanced programming course. The class consisted of 95 mostly junior and senior computer science and computer engineering students. The students were told about pair programming and given the option to use pair programming throughout the quarter. No monitoring of the pairing was performed and no additional information was collected from the students. 44 students elected to complete the programming assignments using pair programming. The remaining 47 students worked alone. In most cases the pairing students worked with the same partner for the entire term.

3.2. Study 2: Randomized Pairing

Data for the second study was collected in winter 2002 on the same course as study 1. The class consisted of 91 mostly junior and senior computer science and computer engineering majors. The study was designed as a one-factor randomized paired comparison. Student volunteers were invited to participate. Unfortunately the voluntary participation rate in the experiment was very low. Only 19 students agreed to participate in the experiment. Participation in the experiment meant being assigned randomly to either a pair programming group (14 students) or a work alone group (5 students). The grouping was designed to balance the number of programs in each, not the number of students. Students in the pairing group would be assigned a partner and would change partners twice during the term. Fairly early in the term two students in the pairing group dropped out of the experiment but remained in the class.

In hopes of encouraging students to pair effectively, the students in the pairing group were asked to evaluate their partner's effectiveness as a partner (not their overall programming ability) *in a manner similar to that of Williams et al.[11]*. This partner evaluation was part of the grade computation for the assignment but done so that the students would not know how any individual partner evaluated them.

We believe that concerns over the partner evaluation system may have contributed to the low participation rate in the experiment. This is supported by responses from the students on a questionnaire at the end of the course. Students were asked if they agreed or disagreed with the statement "I think asking students to evaluate each other and have it affect their grade is a good way to encourage students to pair effectively." Only 1 strongly agreed, 4 agreed, 1 was neutral, 6 disagreed, and 0 strongly disagreed. In addition, 4 students agreed with the statement "Looking back I believe I sometimes gave my partner a higher score than they deserved because I did not want to be responsible for giving them a low grade in the class." The remaining students were neutral (5), disagreed (2), or strongly disagreed (1).

3.3. Study 3: Voluntary Pairing

Data for the third study was collected in spring 2002 on a sophomore level abstract data types course required for all computer science and computer engineering students. There were 102 students that completed at least one of the programming assignments for the class. As with the first study, students in this class were told about pair programming and allowed to use it for the programming assignments. Students could use pair programming for some, all, or none of the assignments. 58 students used pair programming at least once. In most cases when students paired for more than one assignment they worked with the same partner.

In addition to collecting data on the students' performance in the class, students were also asked to complete a questionnaire at the beginning and the end of the class. The questionnaire included questions about the student's perception of their programming ability.

4. RESULTS

In this section we present the results of the various studies. Our focus has been on trying to determine the effect of pair programming on student performance as measured by exam performance and the quality of the programs they submit.

4.1. Final Exam Performance

Table 1 compares the final exam scores for pairing students with non-pairing students in the three studies.

Table 1: Final Exam Scores

Class	Mean	Median	Std. dev	n
Study1 pair	79.6	80.0	12.5	44
Study1 solo	81.9	81.4	12.1	47
Study2 pair	84.7	86.7	9.5	12
Study2 solo	78.2	76.7	8.1	5
Study3 pair	89.8	88.5	8.4	58
Study3 solo	93.3	95.0	11.4	43

In the two larger studies, the average final exam score was higher for the students working individually, however, the difference was never statistically significant.

In Study2 the pair group actually had a somewhat higher final exam average. Because of the small sample size this difference is not significant at the p=.1 level using ANOVA.

For Study3 we obtained data via a questionnaire at the beginning of the class about students' own perception of their programming skill. The students that used pair programming were more likely to have rated their programming ability at the start of the class as poor or average than students working alone. This is consistent with another study that found that students claiming to be strong programmers "liked pair programming the least."[8] This difference was significant at the p=.05 level using a Chi-square test. This would suggest that although self-proclaimed weaker programmers are more likely to use pair programming, in the end their exam scores are not significantly lower then the self-proclaimed stronger programmers.

4.2. Programming Performance

Maybe the final exam does not accurately reflect what students learn (or fail to learn) in the process of doing (or not doing) the programming assignments. Unfortunately, we have not found a practical way to evaluate students' programming ability in a controlled test situation beyond traditional exam questions and a few relatively short "write a program to..." questions. These programs are limited to a few tens of lines of code and must be done with paper and pencil.

Beyond the tests, we have the programs submitted by the students. Without question, some students get more help with their programming assignments than others. This is true whether or not the students use pair programming. In relatively large classes such as the ones we have studied, students generally can get assistance from graduate student teaching assistants or senior undergraduate course assistants. We have found that a persistent weak student can sometimes manage to produce a working program that they could not reproduce unassisted. Pair programming in an open lab provides an additional opportunity for students to have their name on a program that they do not fully understand or could not reproduce on their own. We believe the risk of a few irresponsible students getting undeserved credit for some programming assignments is more than offset by the benefits of pair programming.

Recall that in the previous section we presented data suggesting that students that perceive themselves as strong programmers were less likely to voluntarily use pair programming than self-perceived weak programmers. This would then suggest that the programs produced by the pair programming students should be weaker than those produced by the individual programmers. In fact, the opposite is true.

Table 2 compares the total programming assignment portion of the grade from the same classes discussed in the previous section. Scores are only included for students that took the final exam. In the two larger studies the average program score for students using pair programming is higher than for individuals working alone. These differences are both of practical significance and are statistically significant using ANOVA with $p < .005$. For Study2, although the pair scores are slightly higher, again due to the small size of the study, this difference is not statistically significant.

Table 2: Program Total

Class	Mean	Median	Std. dev.
Study1 pair	89.5	93.3	10.5
Study1 solo	80.2	80	16.6
Study2 pair	90.6	90.0	7.5
Study2 solo	89.7	88.3	10.0
Study3 pair	86.9	90	9.4
Study3 solo	76.7	80.7	20.1

4.3. Blind Evaluation of Program Quality

Some colleagues have raised concerns about the objectivity of the programming assignment evaluations given that all scores reported above were determined by individuals who knew whether a pair or an individual produced the program. To investigate the possibility of grader bias, we took a closer look at the program scores.

4.3.1. Study2 Program Evaluation

As part of our evaluation of the programs for Study2, after the class was over we did an anonymous reevaluation of the programs for the next to last assignment. We felt that the next to last was likely to be most representative. It was a relatively complex program and not so near the end of the course that burnout or end of term conflicts with other classes might affect student performance. Each submission was stripped of any comments or other marks that would indicate who the author or authors were and whether or not an individual or a pair wrote the program. The programs were evaluated on a purely objective functionality scale and a relatively objective style scale. In addition, the programs were ranked from best to worst using a totally subjective evaluation we called the "holistic" evaluation.

The holistic ranking of the programs (done without knowledge of the pairing/non-pairing status of the programs) was obtained by making a single pass over the 11 programs after having computed functionality and style scores for each program. The holistic ranking for the programs was computed 3 times, twice by C. McDowell (the instructor for the class) separated by more than a month, and once by B. Hanks (who has more than 15 years experience as a software professional). Instead of assigning a holistic score, the programs were placed into a ranking as they were read.

The holistic evaluation took in a wide range of program features. For example, one program ended up with a much lower holistic ranking than functionality ranking. Although this program largely satisfied the functional requirements, it was very poorly designed, lacked polymorphism (a key concept for the assignment), had many nearly identical methods, used sequences of *if* statements without an *else* when an *if-else-if-else* was called for, and contained a huge number of fields.

On the other hand, another program received a much higher holistic ranking than functionality ranking. That program elicited very few comments from the reviewers. The program was generally well designed with some weakness in the choice of variable names and in the comments.

Because of the small number of programs (6 pairs and 5 individuals), no real conclusions can be drawn from the numbers; however, we did note one surprising finding. The holistic evaluation resulted in higher average rankings for the pair programs whereas the functionality and style rankings favored the individual programs. Table 3 shows the average rankings for the 11 programs. The differences were not significant but they did cause us to wonder if students working in pairs would be less likely to make the types of errors that resulted in the lower holistic evaluations for the non-pairing programs we observed in this small set.

Table 3: Study2 - Average Rank (11 is highest, 1 is lowest)

	Functionality	Style	Holistic
Pair	5.7	5.9	6.9
Solo	6.4	6.1	4.9

The holistic ranking in the table is the average of the three holistic rankings that were performed. For all three holistic rankings, the pair programs on average moved up more than 1 rank and the solo programs moved down more than one rank. This is shown in

Table 4, where F is the functionality ranking and H1-H3 are the three holistic rankings (H1 and H2 were two separate evaluations by C. McDowell, and H3 was by B. Hanks).

Table 4: Study2 - Average Change in Rank (highest rank is 11 so positive values mean movement to higher rankings)

	H1-F	H2-F	H3-F
Pair	1.3	1.0	1.5
Solo	-1.6	-1.2	-1.8

4.3.2. Study1 Program Evaluation

This discrepancy between the holistic ranking and the more objective (but less thoughtful) functional ranking led us to look at a larger set of programs. For this, one of the authors went back and carefully re-evaluated all submissions of the next to last programming assignment from Study1. The results of this evaluation are shown in Table 5. The table shows the average scores based purely on functionality (0-10) and an overall score (0-10) that took into consideration style, design, and other measures of program quality. Again the pair programming students came out ahead by a statistically significant amount. For both functionally and overall, using ANOVA the differences are significant with $p < .01$.

The number of programs completed by individuals shown in Table 5 exceeds the number of individuals listed in Table 1. The numbers differ because students were allowed to optionally pair on an assignment by assignment basis (most paired consistently), and also because some students did not turn in the assignment.

Although the Study1 data in Table 5 does not corroborate our observed discrepancy between functionality and overall scores seen for Study2, this could be due to our inability to holistically rank the larger set of 69 programs.

Table 5: Study1 Anonymous Program Evaluation

	Mean	Median	Std. dev.	n
Functionality Pair	8.8	9	0.5	16
Functionality Solo	6.6	8	3.1	53
Overall Pair	7.9	8.3	1.7	16
Overall Solo	5.8	7	2.9	53

4.3.3. Related Study Program Evaluation

We also did a similar anonymous evaluation of a random sample of 20 programs from two sections of a beginning programming class reported on in a separate paper[3]. We were trying to determine if there was some identifiable characteristic of programs produced by pairs that was not found in programs produced by individuals (beyond the obvious fact that the programs produced by the pairs were far more likely to function correctly).

The problem with this plan was that the students in the two sections had similar but different programming assignments. The reviewer (one of the authors) did not know which programming assignment was completed by pairs, but was aware that the overall homework scores for the pairing students were significantly higher than for the non-pairing students. If provided with a truly random sample of programs, the reviewer would have quickly noticed that one set contained far more non-working programs, and realized it was the set from the non-pairing class. Instead of abandoning the evaluation, we selected programs from the non-pairing class so that the ratio of fully functioning, partially functioning, and non-functioning programs was the same for both samples. We did this by separating the non-pairing programs into three groups: fully functioning, partially functioning, and non-functioning. We then randomly selected the appropriate number of programs from each group.

The result of this exercise was that the programs in each group were comparable in every respect that we examined. This is remarkable when you consider that 48% of the pair programs were fully functioning but only 12% of the individually written programs were fully functioning. To see if this was an anomaly just for this assignment, we also looked at the percentages of fully/partially/non-functioning programs for the third programming assignment (of 5 assignments total). The data on percentages of fully functioning programs for both assignments 3 and 4 (out of 5 total) is shown in Table 6.

Table 6: Percentage of fully/partially/non-working programs.

	Fully	Partially	Not working
Pair 3rd	61%	13%	26%
Solo 3rd	37%	7%	55%
Pair 4th	48%	32%	20%
Solo 4th	12%	12%	76%

This suggests that students who pair program in an introductory programming class are likely to produce results comparable to those of the best students in a class where students are not allowed to pair.

4.4. Time Spent Programming

The one area where our results differ from some previously published studies is in the area of time spent programming. For Study2 we asked students to report the amount of time they spent completing the programming assignments. We found no significant difference in the amount of time spent working on programming assignments between students working in pairs and students working alone. This contrasts with other reports of pair programming requiring about the same total amount of programmer time for programs developed by pairs and programs developed by individuals (amounting to about one-half as much time spent programming for a member of a pair compared to a programmer working alone)[6, 14]. There are a number of possible explanations for this difference.

We did not provide any type of automated logging tool. The students were simply asked to keep a written log of the time spent and report the total time when they submitted the programs. While there is no reason for us to believe that students intentionally exaggerated the amount of time spent, it is clear that many students did not keep accurate records. We believe that many students did not track time spent while working on their assignments, but instead tried to remember it when they needed to turn in their work. Examples of poor data reporting include situations where partners reported different amounts of time spent together, and time logs where the number of hours spent is always a multiple of 5.

Although the average time reported by an individual working in a pair was 75% of the average time reported by an individual working alone, because of the small sample size (n=17) the

difference is not significant. In addition, the students changed partners every other assignment. Williams et. al report that the difference in total time for pairs relative to individuals decreases as the pair works together longer[9].

5. CONCLUSION

We believe that previous studies and the above data indicate that students should be (at the least) allowed to use pair programming. Previous studies have shown that the use of pair programming in CS1 improves pass rates and retention in the major[4, 5]. Although additional benefits may accrue to faculty and students when pair programming is done in a closed lab, when pairs are assigned, and where students evaluate their partners, simply allowing students to use pair programming results in higher quality programs with no demonstrable disadvantages. The hypothesis that more students pass simply because of higher program scores (thanks to their partner) is inconsistent with our data showing no significant difference in exam scores between pairing and non-pairing students.

We believe that the vast majority of students will learn more working with a partner to create a quality working program than they would struggling on their own to create a non-working program. Our data clearly show that the programs produced by students working in pairs are significantly better than the programs produced by individuals for the same or comparable assignments.

Further study is needed to resolve the inconsistency between our data on the amount of time students spend on programs and that of some other studies. We also need to try and understand how much additional benefit is accrued from some of the more costly aspects of pair programming suggested by others, such as more supervision of the pair programming process, partner evaluations, and changing partners versus working with the same partner. In the mean time, we hope more instructors will take the first step and at least let their students voluntarily pair.

6. ACKNOWLEDGEMENTS

We want to thank J. Fernald, T. Raffill , and P. Tantalo for their help. This work was undertaken as part of C. McDowell's participation in the Carnegie Academy for the Scholarship of Teaching and Learning, and was partially funded by a National Science Foundation grant, EIA-0089989. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation.

7. REFERENCES

[1] Beck, K. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, Reading, Mass, 2000.

[2] Bevan, J., Werner, L. and McDowell, C., Guidelines for the Use of Pair Programming in a Freshman Programming Class. in *15th Conference on Software Engineering Education and Training*, (Covington, KY, USA, 2002), IEEE Computer Society, 100-107.

[3] McDowell, C., Werner, L., Bullock, H. and Fernald, J., The Effects of Pair-Programming on Performance in an Introductory Programming Course. in *33rd SIGCSE Technical Symposium on Computer Science Eduation*, (Northern Kentucky, 2002), ACM Press, 38-42.

[4] McDowell, C.E., Werner, L.L., Bullock, H. and Fernald, J., The Impact of Pair Programming on Student Performance and Pursuit of Computer Science Related Majors. in *International Conference on Software Engineering*, (Portland, Oregon, USA, 2003).

[5] Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C. and Balik, S., Improving the CS1 Experience with Pair Programming. in *34th SIGCSE Technical Symposium on Computer Science Eduation*, (Reno, Nevada, 2003), ACM Press, 359-362.

[6] Nosek, J.T. The Case for Collaborative Programming. *Communications of the Acm, 41* (3). 105-108.

[7] Sanders, D. Student Perceptions of the Suitability of Extreme and Pair Programming. *Computer Science Education* (to appear).

[8] Thomas, L., Ratcliffe, M. and Robertson, A., Code Warriors and Code-a-Phobes: A Study in Attitude and Pair Programming. in *34th SIGCSE Technical Symposium on Computer Science Eduation*, (Reno, Nevada, 2003), ACM Press, 363-367.

[9] Williams, L., Kessler, R.R., Cunningham, W. and Jeffries, R. Strengthening the Case for Pair Programming. *IEEE Software, 17* (4). 19-25.

[10] Williams, L. and Upchurch, R.L., In Support of Student Pair-Programming. in *32nd SIGCSE Technical Symposium on Computer Science Education*, (Charlotte, NC, USA, 2001), 327-331.

[11] Williams, L., Wiebe, E., Yang, K., Ferzli, M. and Miller, C. In Support of Pair Programming in the Introductory Computer Science Course. *Computer Science Education, 12* (3). 197-212.

[12] Williams, L.A. and Kessler, R.R. All I Really Need to Know About Pair Programming I Learned in Kindergarten. *Communications of the ACM, 43* (5). 108-114.

[13] Williams, L.A. and Kessler, R.R., The Effects of "Pair-Pressure" and "Pair-Learning" on Software Engineering Education. in *13th Conference on Software Engineering Education and Training*, (Austin, TX, USA, 2000), IEEE Computer Society, 59-65.

[14] Williams, L.A. and Kessler, R.R. Experiments with Industry's "Pair-Programming" Model in the Computer Science Classroom. *Computer Science Education, 11* (1). 7-20.