Open access • Proceedings Article • DOI:10.1145/1007996.1008043

Program quality with pair programming in CS1 — Source link

Brian Hanks, Charlie McDowell, David Draper, Milovan Krnjajic

Institutions: University of California, Santa Cruz

Published on: 28 Jun 2004 - Technical Symposium on Computer Science Education

Topics: Pair programming

Program Quality with Pair Programming in CS1

Brian Hanks and Charlie McDowell
Computer Science Department
University of California, Santa Cruz
{brianh, charlie}@soe.ucsc.edu

David Draper and Milovan Krnjajic
Applied Mathematics and Statistics Department
University of California, Santa Cruz
{draper,milovan}@ams.ucsc.edu

ABSTRACT

Prior research on pair programming has found that compared to students who work alone, students who pair have shown increased confidence in their work, greater success in CS1, and greater retention in computer-related majors. In these earlier studies, pairing and solo students were not given the same programming assignments. This paper reports on a study in which this factor was controlled by giving the same programming assignments to pairing and solo students. We found that pairing students were more likely to turn in working programs, and these programs correctly implemented more required features. Our findings were mixed when we looked at some standard complexity measures of programs. An unexpected but significant finding was that pairing students were more likely to submit solutions to their programming assignments.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: Computer Science Education

General Terms

experimentation, measurement

Keywords

CS1, pair programming, collaboration, student perception

1. INTRODUCTION

Pair programming [8] transforms what has traditionally been a solitary activity into a cooperative effort. While pair programming, two software developers share a single computer monitor and keyboard. One of the developers, called the *driver*, controls the computer keyboard and mouse. The driver is responsible for entering software design, source code, and test cases. The second developer, called the *navigator*, examines the driver's work, offering advice, suggesting corrections, and assisting with design decisions. The developers switch roles at regular intervals. Although role switching is an informal process, a typical interval is 20 minutes.

In an experiment conducted during the 2000–01 academic year, students in three sections of a CS1 course (introductory programming) at UC Santa Cruz pair programmed. Students in a fourth section of the course worked alone. Significant findings from this research [6] include: (1) a larger percentage of paired students passed the course; (2) students who paired had greater confidence in their work, enjoyed their work more and were more satisfied with the programming process; (3) students who paired were more likely to attempt CS2 and passed it at equal rates; and (4) a greater percentage of students who paired were still in a computer-related major one year later.

One uncontrolled variable in the above study was the programming assignments; students who paired were not assigned the same programs as those who worked alone. So that we could better compare the performance of pairing and solo students, in the work reported on here we gave the same programming assignments to pairing students in the winter 2003 offering of the course that had been given to the non-pairing students in the spring 2001 course (the solo course from the 2000–01 study). Due dates were set so that students in both courses had the same amount of time available to complete their work. We expected that students who paired would produce higher quality programs than those who worked alone. We also wanted to confirm the earlier findings that paired students have greater confidence and are more satisfied in their work. Our study design is observational (i.e., students were not assigned to the pairing or non-pairing "treatments" at random), but we found no major relevant differences, in variables correlated with programming ability, between the pairing and non-pairing students at baseline.

We hypothesized that pairing students would

H1 perform better in terms of the number of features successfully implemented;

H2 produce programs that are shorter and less complex;

H3 produce programs that show a better understanding of the basic programming concepts discussed in lecture; and

H4 be more confident in their solutions and more satisfied with the programming process.

We did not give the students in these two classes the same

ITiCSE'04, June 28–30, 2004, Leeds, United Kingdom.

examinations. Therefore, we did not compare student performance on exams or on course grades.

2. PRIOR RESULTS

In addition to the findings described above, there is substantial evidence that pair programming provides significant pedagogical benefits.

Williams [9] reported that advanced undergraduate students who paired produced programs that successfully passed more test cases than students who worked alone. These higher quality programs took only slightly more total programmer time to develop. Other evidence that student pairs create higher quality programs includes reports that student pairs produce programs that are shorter, and thus easier to maintain [1], or are better designed [5, 8].

For CS1 students, pair programming improves their laboratory experience [7]. Pairing students are able to answer more of their own questions, allowing the teaching assistant to focus on more substantive issues. Pairing students in these labs also have fewer "give-ups," in which they have a question but give up because the teaching assistant is busy with other students. The pairing students in this study also did well in CS2. They passed CS2 at the same rates as, or better rates than, the non-pairing CS1 students, even though all students worked alone in CS2.

3. PROGRAM EVALUATION

In the winter 2003 term, we gave pairing students in a CS1 course the same set of programming assignments that had been given to non-pairing students in spring 2001. For each of five assignments, we have approximately 100 programs completed by non-pairs and 25 completed by pairs. We evaluated the last three of these programs to see if there are any qualitative differences between the sets of programs. We decided not to evaluate the first two programming assignments, as we felt that they were so simple that there would not be any pertinent quantifiable differences between the programs produced by the pairing and solo students.

Programs were evaluated using both objective and subjective measures. Objective measures included the number of features that the students correctly implemented, the length of the programs, and the cyclomatic complexity number (CCN) of the programs. Subjective measures included the use of meaningful identifiers, well-organized methods, appropriate indentation and whitespace, and use of booleans instead of two-valued integers as control predicates.

We used the open-source tool JavaNCSS [3] to calculate source code metrics for the programs: the number of non-comment lines of code, the number of methods, the length of the longest method, the average method length, the average method complexity, and the complexity of the most complex method. Program and method lengths are measured in non-comment source lines of code. JavaNCSS uses McCabe's cyclomatic complexity number [4] as its complexity measure.

3.1 Program Three

For programming assignment three, students were asked to write a program to play the card game blackjack. Students had just learned about methods and were expected to write them in this assignment. They were given classes that implemented the deck of cards.

For this assignment, we scored programs by counting the number of features that the students correctly implemented. From this score we subtracted the number of defects that the program exhibited, and called the resulting variable DIFF3. There were twenty features of interest, so the total score ranged from 0 to 20. A program could not get a negative score, because the defects were related to an attempt to implement the feature. For example, programs that did not compile exhibited none of the defects, and scored 0.

The sample size, mean, and standard deviation of DIFF3 for the paired students were $n_P = 24$, $\bar{P} = 13.67$, and 5.30, respectively. For the solo students, these values were $n_S = 105$, $\bar{S} = 11.10$, and 7.51. The paired mean was 23% higher than the solo mean, a difference which we regard as significant in practical terms. In a Bayesian analysis of these data [2], focusing on posterior probability distributions for means in the populations of students exchangeable with (similar to) those who took part in our study (and using diffuse prior distributions for those means), the posterior probability that the population difference ($\mu_P - \mu_S$) between means on DIFF3 is positive is 98%, implying posterior odds of 39.8 to 1 that pairing represents an improvement not just in our sample but also in the underlying populations. (See the Appendix for links between the Bayesian findings in this paper and corresponding classical results based on p-values.)

An examination of the paired and solo distributions on this variable revealed that much of this difference arose because pairing noticeably helped the students avoid getting a 0: the rate of 0 scores in the paired group (8.3%) was 56% lower than the corresponding rate for solo students (19.1%). The posterior probability that the population difference (solo – paired) between rates of 0 scores is positive was 94%; this corresponds to posterior odds of 16.2 to 1 that pairing yielded an improvement. A score of 0 was almost always due to the program failing to compile. That is, pairing students were less likely to turn in programs that did not compile.

We also used JavaNCSS to calculate size and complexity measures for all programs that received a score of 12 or more. Twelve was selected as the cutoff point for this exercise because there was a break in the score distribution at this point, and because programs that scored at least that much were mostly functional. Table 1 summarizes these measures (sample sizes in this table were $n_P = 20$ and $n_S = 64$, except that one extremely outlying observation was set aside in the program length analysis). For all six variables in this table the mean with pair programming was smaller than with solo programming (the relative decreases from the solo means ranged from 4.7% to 22.7%); the posterior probabilities that the population means under pair programming are smaller than under solo programming ranged from .65 to .94 (with corresponding posterior odds ranging from 1.8:1 to 19.6:1). Taken together the results in Table 1 offer moderate support for the hypothesis that pairing students produce programs that are shorter and less complex, at least for assignments like our program three.

3.2 Program Four

The fourth programming assignment asked students to write a program that implemented a simple dice game. This assignment required students to use one-dimensional arrays.

We evaluated this assignment against seventeen test criteria. Only a small percentage of the programs were fully functional. The majority exhibited serious problems. Only

		Mean/SD	$\frac{\bar{P}-\bar{S}}{S}$	$(\mu_P < \mu_S)$ Prob.	Odds
Program Length	P	88.4/15.3	−.068	.91	10.1
	S	94.8/25.9			
# of Methods	P	5.0/2.2	−.083	.76	3.1
	S	5.5/3.4			
Avg Meth Length	P	21.6/10.7	−.227	.95	19.6
	S	27.9/23.8			
Max Meth Length	P	46.9/22.9	−.047	.65	1.8
	S	49.2/26.9			
Avg Meth CCN	P	5.5/2.1	−.188	.94	16.7
	S	6.8/5.2			
Max Meth CCN	P	11.0/4.4	−.075	.77	3.3
	S	11.9/5.6			

Table 1: Program 3 Complexity Measures

17 out of 127 programs (13.4%) received a score of 15 or more on our scale, while 67 of the programs (52.8%) scored 7 or less.

Using statistical analyses like those on program three, for program four we found no meaningful differences between the programs produced by pairs and non-pairs on any of the subjective or objective measures. Of course, since many of the programs did not work, this evaluation may have had limited value.

We have come to believe that this assignment was particularly difficult for the students. In the spring 2001 class, the average grade on this assignment for programs that were turned in was 2.94 on a 5 point scale, while the lowest average grade on the other assignments was 3.47. In the winter 2003 class, the average grade on this assignment was 3.44, while the lowest average grade on the remaining assignments was 4.30. (Students in both classes could get more than 5 points by doing extra-credit work.) The students were also less confident in their solutions to this assignment (as discussed in Section 5.1 below).

We are concerned that this assignment was so challenging for the students that they spent all of their effort just trying to get the program to run. This prevented them from focusing on other aspects of program development, such as the use of meaningful variable names and well-organized control flow.

3.3 Program Five

The fifth programming assignment asked students to write a program to implement a text-based version of the Minesweeper game. Students needed to use two-dimensional arrays on this assignment.

As with assignment three, we scored these programs by counting the number of features that the students correctly implemented. From this score we subtracted the number of defects that the program exhibited, and called the resulting variable DIFF5. There were nine features of interest, so the total DIFF5 score ranged from 0 to 9. A program could not get a negative score, because the defects were related to an attempt to implement the feature. For example, a program that did not compile did not exhibit any of the defects, and received a score of 0.

The sample size, mean, and standard deviation of DIFF5 for the paired students were 24, 5.89, and 2.91, respectively. For the solo students, these values were 89, 5.01, and 3.43. The paired mean was 17% higher than the solo mean, a difference which we regard as significant in practical terms. In a Bayesian analysis of these data, similar to the one described in Section 3.1, the posterior probability that the population difference (paired – solo) between means on DIFF5 is positive is 89%, implying posterior odds of 8.3 to 1 that pairing led to an improvement.

An examination of the paired and solo distributions on this variable revealed that most of this difference again arose because pairing noticeably helped the students avoid getting a 0: the rate of 0 scores in the paired group (8.3%) was 61% lower than the corresponding rate for solo students (21.4%). The posterior probability that the population difference (solo – paired) between rates of 0 scores is positive was 97%; this corresponds to posterior odds of 28.6 to 1 that pairing yielded an improvement. As with program three, a score of 0 was almost always due to the program failing to compile; pairing students were substantially less likely to turn in programs on assignment five that did not compile.

		Mean/SD	$\frac{\bar{P}-\bar{S}}{S}$	$(\mu_P > \mu_S)$ Prob.	Odds
Program Length	P	190.6/63.6	+.380	.99	577.2
	S	138.1/38.1			
# of Methods	P	13.1/5.1	+.137	.86	6.2
	S	11.5/3.4			
Avg Meth Length	P	14.6/4.7	+.271	.99	119.0
	S	11.5/2.6			
Max Meth Length	P	45.8/25.6	+.569	.99	85.8
	S	29.2/16.8			
Avg Meth CCN	P	5.8/2.6	+.345	.98	58.4
	S	4.3/1.0			
Max Meth CCN	P	18.2/15.3	+.565	.94	15.1
	S	11.7/8.8			

Table 2: Program 5 Complexity Measures

As with program three, we used JavaNCSS to calculate size and complexity measures for the programs that scored seven or more (i.e., were considered to be mostly working); Table 2 summarizes the findings (the sample sizes in this table were $n_P = 14$ and $n_S = 44$). Here the results were surprising: the paired programs were 14% to 57% longer and more complex than those produced by students working alone (the posterior probabilities and odds that $\mu_P > \mu_S$ ranged from .86 to .99 and from 6.2:1 to 577.2:1, respectively). This is inconsistent with findings of previous investigators that pairing students produce shorter programs [1] that are better designed [8, page 38], and stands in contrast to the results in Section 3.1. Further investigation is needed to understand what aspects of programs three and five have led to these sharp differences.

4. HOMEWORK SUBMISSION RATE

The above analysis was performed on homework assignments that were turned in by students. Unfortunately, some students don't do their homework.

Table 3 lists the number of students who turned in solutions to the homework assignments. Only the 112 solo students and 50 pairing students who took the final exam are included in this table. (Five of the solo students were

enrolled in the pairing class, but worked by themselves for the entire term. For each assignment, one out of the five students did not submit a solution.) Pairing students turned in solutions to their programming assignments at noticeably higher rates ($\hat{R_P}$) than solo students ($\hat{R_S}$), with the differences ranging from 7.6 to 14.8 percentage points; the posterior probabilities that the population differences ($R_P - R_S$) between submission rates for paired and solo students are positive ranged from .94 to over .999.

		No. Sub.	$\hat{R}$	Diff. $(\hat{R}_P - \hat{R}_S)$	$(R_P > R_S)$ Prob.	Odds
HW3	P	48	.96	+.076	.94	16.9
	S	99	.88			
HW4	P	50	1.0	+.120	.998	499.0
	S	99	.88			
HW5	P	48	.96	+.148	.998	525.3
	S	91	.81			
Total	P	146	.97	+.113	> .999	> 999.9
	S	289	.86			

Table 3: Submission Rate

We believe that it is especially noteworthy that pairing students turned in their homework at higher rates than non-pairing students on the fourth and fifth assignments. The students disliked the fourth assignment and found it very challenging. The fifth assignment was due during the last week of the term, when students have many conflicting due dates in their other courses.

We are very encouraged that students who pair attempt the homework assignments at very high rates, even when they are frustrated or feel overwhelmed by their workload. Perhaps pairing students feel pressure not to let their partner down, or they encourage and motivate each other when they would otherwise give up. Pairing students' increased confidence and satisfaction may also be playing a role here.

5. CONFIDENCE AND SATISFACTION

We have previously reported [6] that students who paired in CS1 had greater confidence in their work and had greater satisfaction with the programming process than students who worked alone. As noted above, one uncontrolled variable in our earlier study was the programming assignments; students who paired were not assigned the same programs as those who worked alone. We have controlled this variable in the study reported here.

Students in the earlier study were asked to respond to questions regarding their confidence and satisfaction on each programming assignment. These questions are reproduced in Table 4. We asked the students in the 2003 paired class to answer the same questions. Unfortunately, we do not have their responses to these questions for the third assignment, so it is not included in the analysis in this section.

5.1 Confidence

On every programming assignment except HW4, students who paired were more confident in their solutions than those who worked alone (see Table 5), by margins that are significant both practically and statistically. Overall, aggregating across all assignments, confidence was 12.2% higher in the paired group on average, and the posterior probability that the population mean difference ($\mu_P - \mu_S$) in confidence is positive exceeded .999. These results strengthen our earlier findings that students who pair are more confident than students who work alone.

Confidence	On a scale from 0 (not at all confident) to 100 (very confident), how confident are you in your solution to this assignment?
Satisfaction (pairs only)	How satisfied are you with the way you and your partner worked together on this assignment? (1 = very dissatisfied, 7 = very satisfied)
Satisfaction (non-pairs only)	How satisfied are you with how you spent your time on this assignment? (1 = very dissatisfied, 7 = very satisfied)

Table 4: Questions asked about each program

		n	Mean/SD	$\frac{\bar{P}-\bar{S}}{\bar{S}}$	$(\mu_P > \mu_S)$ Prob.	Odds
HW1	P	46	90.1/15.6	+.158	.999	> 999.9
	S	93	77.8/31.0			
HW2	P	31	87.4/20.5	+.218	.999	728.3
	S	90	71.7/35.1			
HW4	P	21	55.1/33.5	−.203	.05	0.051
	S	70	69.1/35.1			
HW5	P	24	81.0/17.3	+.147	.97	34.9
	S	68	70.6/33.8			
Total	P	122	81.6/24.4	+.122	.999	941.5
	S	321	72.7/33.7			

Table 5: Student Confidence

5.2 Satisfaction

This study confirmed our earlier findings that students who pair are more satisfied with the way they work (see Table 6). Pairing students were more satisfied on every program, by margins of 13.8% to 28.4% on average. Overall, aggregating across all four assignments, on average satisfaction was 22.0% higher for paired students, and the posterior probability that the population aggregate mean difference ($\mu_P - \mu_S$) was positive again exceeded .999. These results should be viewed with a bit of caution, because the paired and non-paired students were not asked identical questions. We are encouraged, however, that the results here strengthen our earlier findings.

6. CONCLUSIONS

For two of the three assignments we studied our analysis confirms hypothesis H1, that pairing students would perform better in terms of the number of features successfully implemented. We did not detect any differences between the two groups on one of the programming assignments.

We were not able to uniformly confirm our hypotheses H2 and H3, as we had mixed results. On the third assignment, pairs wrote programs that were shorter and less complex; on the fourth assignment, there were no significant differences

		n	Mean/SD	$\frac{\bar{P}-\bar{S}}{S}$	$(\mu_P > \mu_S)$ Prob.	Odds
HW1	P	46	6.00/1.43	+.213	> .999	> 999.9
	S	92	4.95/1.67			
HW2	P	31	5.94/1.73	+.284	.999	> 999.9
	S	90	4.62/1.69			
HW4	P	21	5.43/1.57	+.138	.94	17.0
	S	70	4.77/1.93			
HW5	P	25	5.88/1.39	+.204	.997	365.7
	S	68	4.88/1.87			
Total	P	123	5.86/1.52	+.220	> .999	> 999.9
	S	320	4.80/1.78			

Table 6: Student Satisfaction

between the two groups on any of the complexity measures; and on the last assignment, pairs wrote programs that were longer and more complex. There is an evident trend for the length and complexity of programs produced by the pairs to increase as the difficulty of the assignments increased; this trend was not uniformly present for the solo programmers. There was no evidence that either group of students had a better understanding of basic programming concepts.

We were able to confirm that paired students are more confident in their programming solutions and are more satisfied with the programming process than students who work alone. This finding strengthens our earlier results, since students in the pair and solo groups worked on the same assignments. Thus, hypothesis H4 is confirmed for confidence and satisfaction.

We believe that one of the most significant findings of this study is the increased homework submission rates observed in pairing students. Learning to program is very difficult for many students, and the best way to learn programming is by writing programs. It appears that pair programming encourages students to work on their programming assignments. It seems likely that these students are learning more, because they are actually attempting the homework.

As discussed earlier, students who paired were more likely to turn in programs that compiled. Combined with the greater submission rate, this shows that the pairing students were much more successful at overcoming the hurdles that frustrated solo students.

Although we cannot confidently state that pairing students write programs that are better designed and show a greater understanding of basic programming concepts, we believe that the benefits of pair programming outweigh its costs. Students who pair write programs with greater functionality, are more confident in their work, are more satisfied with the programming process, and are more likely to work on their programming assignments. These findings add to the growing body of evidence that pair programming increases student success in computer science courses.

7. ACKNOWLEDGMENTS

This work was funded by 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.

Appendix

The explicit link between the Bayesian analyses presented here and classical analyses based on significance testing is as follows: a posterior probability of q that $(\mu_P - \mu_S)$ is positive (given the data and diffuse prior distributions) corresponds to a p-value of $(1-q)$ when testing the null hypothesis that $\mu_P < \mu_S$. Like many others (see, e.g., the references in [2]) we find the Bayesian analysis more directly interpretable.

8. REFERENCES

[1] A. Cockburn and L. Williams. The costs and benefits of pair programming. In G. Succi and M. Marchesi, editors, *Extreme Programming Examined*, pages 223–247. Addison-Wesley, 2001.

[2] A. Gelman, J. Carlin, H. Stern, and D. Rubin. *Bayesian Data Analysis*. Chapman and Hall CRC, New York, second edition, 2003.

[3] C. Lee. JavaNCSS - a source measurement suite for Java. `http://www.kclee.com/clemens/-java/-javancss/`, current September 2, 2003.

[4] T. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320, Dec. 1976.

[5] C. McDowell, B. Hanks, and L. Werner. Experimenting with pair programming in the classroom. In *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education*, June 30–July 2, 2003.

[6] C. McDowell, L. Werner, H. Bullock, and J. Fernald. The impact of pair programming on student performance, perception and persistence. In *Proceedings of the International Conference on Software Engineering (ICSE 2003)*, pages 602–607, May 3–10, 2003.

[7] N. Naggapan, L. Williams, E. Wiebe, C. Miller, S. Balik, M. Ferzli, and J. Petlick. Pair learning: With an eye toward future success. In *Extreme Programming and Agile Methods - XP/Agile Universe 2003*, number 2753 in LNCS, pages 185–198. Springer, 2003.

[8] L. Williams and R. Kessler. *Pair Programming Illuminated*. Addison-Wesley, 2002.

[9] L. A. Williams. *The Collaborative Software Process*. PhD thesis, University of Utah, 2000.