The Benefits of Pairing by Ability

Grant Braught, John MacCormick, and Tim Wahls
Dickinson College, Carlisle, PA 17013
{braught, jmac, wahlst}@dickinson.edu

ABSTRACT

An analysis of data from 259 CS1 students is performed to compare the performance of students who were paired by demonstrated ability to that of students who were paired randomly or worked alone. The results suggest that when given individual programming tasks to complete, lowest-quartile students who were paired by ability perform better than those who were paired randomly and those who worked alone.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: Computer science education; D.2.3 [**Software Engineering**]: Coding Tools and Techniques

General Terms

Experimentation, Human Factors

Keywords

Pair programming, pairing methodology, collaborative learning

1. INTRODUCTION

Controlled experiments on the effects of pair programming have been reported from a variety of institutions and for a wide range of different courses. These studies present convincing evidence that students who pair program are more likely to successfully complete the course [1, 8, 11], have increased confidence in their solutions [2, 9], are more likely to continue with computer science [9], and develop better individual programming skills [1]. Additionally, some studies have shown that pair programming is particularly beneficial for female and minority students [3, 16, 18], and also for students with lower SAT scores [1].

Numerous studies have also investigated the factors that affect pair compatibility when pair programming is used. The majority of these studies paired students randomly and then used quantitative and/or qualitative methods to identify correlations between the students' ratings of their pairings and factors such as ability, perceived ability, confidence, personality type, work ethic, gender and ethnicity. By far the most common conclusion is that pairs are most likely to be compatible when students are paired with a partner who they perceive as being of equal or higher ability [4, 7, 6, 10, 17]. The results are considerably less clear for pairs with similar actual ability (as measured by some combination of GPA, SAT/GRE and midterm exam scores) — some studies report a positive correlation with pair compatibility [6], while others could not establish a consistent correlation [7, 17]. Several of the studies included personality tests such as the Myers-Brigs Type Indicator, Felder-Silverman Learning Style or Keirsey Temperament Sorter and concluded that pairs comprising students with different personality traits were more likely to be compatible than homogeneous pairs [7, 13, 17]. A number of the studies also examined the effects of self-esteem or confidence on pairing, producing mixed results. One study found that pairing students with similar self-esteem did not influence the likelihood of pair compatibility [7], another found that the most confident students reported enjoying pairing more than less confident students [5], while another found the opposite [14]. One study that considered gender and ethnicity found that for female and minority students, being paired with another female or minority student was positively correlated with pair compatibility [6].

The factors affecting pair compatibility have been investigated quite extensively, but surprisingly few studies have attempted to directly measure the effects of the pairing methodology on student achievement. Thomas, Ratcliffe and Robertson [14] studied the effects of pairing students by their confidence in their own coding skills. Pairs in which both students had similar confidence (e.g. high/high or low/low) received the highest scores on assignments, and further significant differences were not found between the scores of low/low pairs and high/high pairs. They also report that high/low pairs received the lowest scores. Sfetsos *et al.* [13] created pairs with similar and dissimilar temperaments, as measured by the Keirsey Temperament Sorter. They found that heterogeneous pairs communicated more often, produced better designs and wrote code that passed more acceptance tests than homogeneous pairs.

Both of the studies just described measure the impact of the pairing methodology on the quality of the work produced by the *pairs*. One might also ask how different pairing

SIGCSE'10, March 10–13, 2010, Milwaukee, Wisconsin, USA.

methodologies impact the *individuals* within the pairs. In the remainder of this paper we present evidence suggesting that when students are paired with a partner of similar ability, the individuals within the pair each learn at least as much as if they had either worked with a random partner or worked alone. The evidence also suggests that for the bottom quartile of students, pairing by similar ability may result in increased learning for the individuals in the pair. Some limitations of the dataset prevent us from drawing firm conclusions, but we do believe our analysis provides useful insights into the benefits of by-ability pairings and will motivate more carefully controlled studies in the future.

2. DESCRIPTION OF THE DATASET

Our dataset consists of 13 sections of a one-semester, entry-level Java programming class, taught by four different instructors between 2005 and 2008. The content and format of the class, including homework and programming assignments, was virtually identical in each section. Only students who completed the course are included in the dataset (that is, we eliminate students who withdrew, but not those who failed). Enrollment in each section was capped at 24 students. An average of 20 students completed each section, resulting in a dataset of 259 students.

Students are graded on four different types of work: (i) weekly written homework exercises; (ii) weekly programming assignments; (iii) three written exams; (iv) two programming exams. Because the programming exams are of special importance in our analysis, we describe them in some detail here. The required task in a programming exam is to complete the fields and method bodies of the skeletons of one or more Java classes, and also to write unit tests for each method. The provided skeletons contain the method signatures and detailed JavaDoc comments. During a programming exam, students work individually in a proctored room, receive essentially no assistance, and have a fixed amount of time (two hours) to complete the exam. Grading of the exam is mostly automatic (via Web-CAT [15]), although instructors can deduct a small number of points for poor programming style. The automatic grading assesses formatting and documentation, runs reference tests to verify correctness, and checks for code coverage of unit tests. Although the actual content of the programming exams changed each semester, considerable efforts were made to keep the structure, length, difficulty, and grading of these exams as uniform as possible over the study period.

In contrast to the programming *exams*, which were administered uniformly over all sections, the weekly programming *assignments* were administered quite differently in different sections. In particular, three different approaches were used: (i) *ability pairs* – students work on programming assignments in pairs assigned by ability, with students of similar ability (as measured by overall performance in the course to date) working together; (ii) *random pairs* – students work on programming assignments in pairs assigned randomly; and (iii) *no pairs* – students work on programming assignments individually. In the two pairing approaches, students are required to rotate the "driver" (i.e. typist) role every 15 minutes during the 2-hour laboratory session. Students were not aware of the criteria for assigning pairs. Of the 13 sections in the dataset, 7 paired students by ability (142 students), 2 paired students randomly (41 students), and 4 used no pairs (76 students).

We will refer to students as being "trained with" one of the three approaches, and for additional brevity will use phrases like "ability-pair students" to refer to students trained with ability pairs. The main purpose of this paper is to assess the effect of these three training strategies on student outcomes.

3. ANALYSIS

We are interested in the hypothesis that students trained with ability pairs achieve better individual outcomes than students trained with random pairs or no pairs — especially for less able students. The "outcomes" will be the scores achieved by students on the final programming exam of the semester, given in the 14th week. It is important to note that this outcome measures a student's skill when programming *alone*, regardless of which of the three training approaches was used by the student in previous assignments. Previous work [1] has shown that ability pairing benefits the lowest quartile of students, compared with using no pairs. We would like to investigate whether the same is true for random pairings. Therefore, our hypothesis tests will focus on the lowest quartile of students, as measured by their programming exam scores. For concreteness, the hypothesis test comparing random and ability pairs is described in detail; the test comparing no pairs with ability pairs is perfectly analogous. Specifically, the null hypothesis for random pairs will be, "the 25th percentile of programming exam scores for students trained with random pairs is the same as for students trained with ability pairs."

The test statistic used in this experiment, denoted n_{25}, is very simple: it is the number of students trained with random pairs who scored less than the 25th percentile of the scores for students trained with ability pairs. Figure 1 shows the numerical results for our dataset. The 25th percentile of scores for the 142 students trained with ability pairs is 82/100 (i.e. 25% of the 142 students scored less than 82 on the programming exam — this is the intersection of the ability-pair line and the dashed horizontal line in the figure). Of the 41 students trained with random pairs, 14 scored less than 82 (i.e. are left of the vertical line) on the programming exam. Therefore, the test statistic is $n_{25} = 14$. The conventional approach is to compute the p-value of the test statistic, and reject the null hypothesis if the p-value is lower than a given level of significance, such as 0.05 or 0.01.

How can we convert this test statistic into a p-value? Intuitively, if the null hypothesis were true and the 25th percentile for random-pair students was the same as for ability-pair students, then approximately 10 (i.e. one quarter of 41) of the random-pair students should have scored less than 82. The fact that considerably more than 10 students (actually 14) scored below 82 immediately raises suspicions that the null hypothesis may not hold. To quantify this suspicion and obtain a p-value, we compute the probability that a value of $n_{25} \geq 14$ would have occurred if the null hypothesis were true. This probability can be calculated as the tail of a binomial distribution: $p = \sum_{k=n_{25}}^{N} \binom{N}{k} \pi^k (1-\pi)^{N-k}$. Here, $N = 41$ is the number of random-pair students and $\pi = 0.25$ is the value of the percentile used in the null hypothesis. Each term in the sum is the chance that k of N students would score less than 82 if, as stated by the null hypothesis, the chance of doing so is 25%.

For this experiment $p = 0.12$, which means that there is a 12% probability that a discrepancy of at least this amount (between the expected number of students scoring below 82,

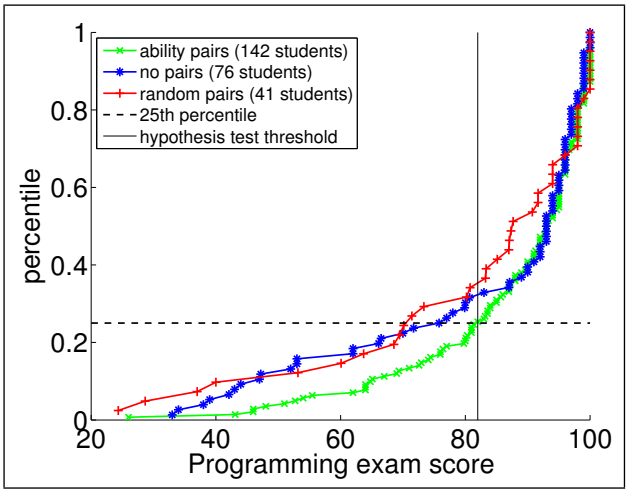

Figure 1: The method of pairing students for assignments has potentially significant effects on the performance of the bottom quartile of students. Below the 25th percentile (dotted line), the difference between ability pairs and the other two methods has mild statistical significance (p-values ranging between 0.02 and 0.12).

and the actual number doing so), would have arisen if the null hypothesis were true. It is conventional in statistical analyses to reject the null hypothesis only if the p-value is considerably lower than this (say, 0.01 or 0.05). However, given the small amount of data available for this study, we regard the p-value of 0.12 as providing some evidence that the null hypothesis may be false, and that more extensive studies of such hypotheses would be worthwhile. On the other hand, it should also be noted that p-values in post-hoc studies such as this one are frequently more extreme than they would have been in a controlled study, so some care must be taken in assessing the significance of this result.

Some additional remarks on our choice of statistical tests are in order. First, a more conventional way to test the difference between two distributions (such as the distributions of scores for ability-pair students and for random-pair students) would be to apply the Kolmogorov-Smirnov (KS) test. But because we have a priori reasons to believe the differences will be concentrated in the lower quartile [1], KS is less than ideal for this application. Indeed, KS has lower power than the hypothesis test described above, and yields a p-value of 0.39 for the hypothesis that the two distributions coincide. A p-value this high provides no evidence for rejecting the null hypothesis. Second, the above test based on binomial probabilities assumes that the 25th percentile of ability-pair scores (82/100 in this case) is the true 25th percentile for the population of students trained by ability. In fact, 82/100 is merely an estimate for this quantity based on the 142 ability-pair students in our dataset. A more sophisticated test taking this uncertainty into account could be applied. However, given the relatively high number of ability-pair students we believe the simpler approach taken here still yields useful results.

3.1 Tests related to the main hypothesis

The previous subsection described the concrete details of one particular hypothesis test, comparing the lowest quartile of students trained with ability pairs and students trained with random pairs. Naturally, we can perform the same test comparing ability-pair students with no-pair students, and both tests can be done for any desired percentile instead of the 25th percentile discussed so far. The following table gives the p-values for some representative points in this space of hypothesis tests, using the same data from Figure 1.

percentile	random pairs	no pairs
10%	0.11	0.02
15%	0.04	0.03
25%	0.12	0.12
50%	0.17	0.36

It should be clear that below the 25th percentile, there are potentially significant differences between training with ability pairs and training with the other two approaches (random pairs and no pairs). Any differences at the 50th percentile are much less clear. Hopefully it is also clear from these results that we have not simply cherry-picked one particular percentile (in this case, the 25th percentile) for our discussion because it happens to yield statistical significance. In fact, we see much stronger significance at the 10th and 15th percentiles — but the very small number of students in the tails of these distributions makes us reluctant to draw conclusions about this.

3.2 Other hypotheses

The main thrust of our analysis has been to test the hypothesis that pairing by ability improves the performance of weaker students on solo programming tasks, but our dataset is suitable for testing numerous other interesting hypotheses. For example, given the extensive evidence in the literature on the educational benefits of pair programming, one might expect to see a significant difference between students trained with random pairs and students trained with no pairs. However, a quick glance at Figure 1 should be sufficient to demonstrate that our dataset cannot validate this hypothesis. It may well be true that random pairings generally produce better outcomes than no pairings at all, but in our data the bottom quartile of random-pair students and no-pair students have very similar performance on the programming exam.

We are also interested in whether the choice of pairing method affects student performance on all individual activities in the course. To investigate this, we re-ran the analysis of Section 3, substituting the students' score on all solo assignments (homework, programming exams, and written exams) for the final programming exam score. The results using this "solo activity score" are shown in Figure 2. Based on an informal visual assessment of the graph, there does appear to be a noticeable difference between ability pairs and the other two approaches in the lowest quartile — though perhaps not as striking as the difference for the programming exam shown in Figure 1. In fact, if we compute the p-value of the 25th percentile statistic n_{25} defined above, we find the difference is somewhat significant for no-pair students (p-value 0.12), but insignificant for random-pair students (p-value 0.45).

Finally, one might consider the effect of pairing method on student performance in pair programming assignments themselves. Again, a very similar analysis can be performed, this time measuring student outcomes by their total score on all pair programming assignments, termed a student's

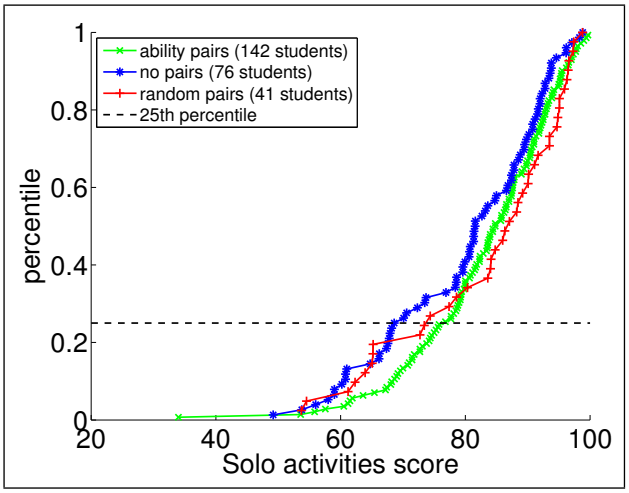

Figure 2: Pairing method may have a mild effect on the lowest quartile of students. In contrast to Figure 1, which measured student performance on a programming exam only, this figure shows their performance on all individual activities including homework, programming exams, and written exams. In the lowest quartile, there do appear to be noticeable differences in performance.

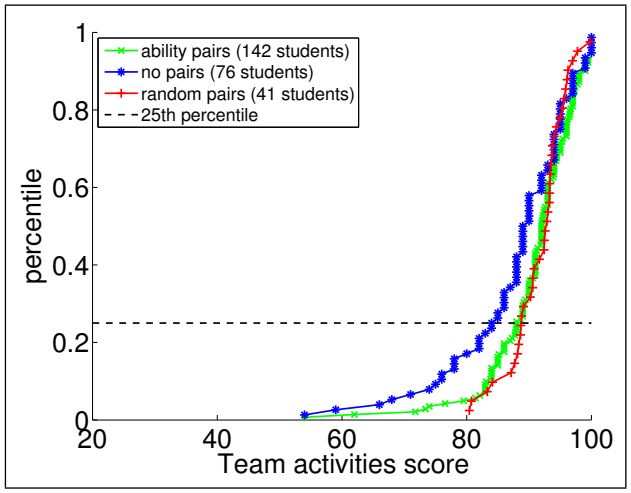

Figure 3: In the lowest quartile, there is overwhelming statistical evidence that students working alone achieve lower scores on a given set of assignments than students working in pairs (whether random or by ability) on the same set of assignments. In other words, below the dotted line representing the lowest quartile, the red and green lines are statistically indistinguishable but the blue line exhibits a highly significant difference.

"team" score. (Obviously, for students trained with no pairs, there are no pair programming assignments. But we can still collect the scores of the no-pair students on the assignments that were done in pairs by students in other sections. This results in a comparable number that is still referred to as the "team" score for no-pairs students.) Figure 3 shows the results. Here we see that outcomes for ability pairs and random pairs are virtually indistinguishable (p-value 0.96), whereas a highly significant difference does exist in the lowest quartile between no-pair students and the other two approaches (p-value $< 10^{-3}$ for no-pair vs random, and for no-pair vs ability). In fact, there is high significance for differences at the 50th percentile too (p-value ≈ 0.014 in both cases).

The stark difference in team scores for no-pair and paired approaches is not at all surprising. The instructors in our study have observed strong anecdotal evidence that students are much less likely to give up on a challenging assignment when working with someone else, and this tends to dramatically improve the performance of the bottom quartile.

3.3 Confounding factors

We believe our dataset provides useful and novel insights, but it possesses some limitations that should be acknowledged clearly. First, this is a partially post-hoc study, in the sense that the random pairing technique was not part of the original design. The majority of the data was collected from a carefully-controlled study that compared pair programming using by-ability pairs with individual programming [1]. The same course was subsequently taught twice using random pairings, resulting in a dataset with potentially useful information about different pairing strategies. Second, the pairing strategies themselves were not applied with perfect consistency: instructors used different combinations of automatic grade-based scripts and their own subjective impressions in assigning pairs. In any case, ability pairs are not used early in the semester when the instructors have no grades or impressions on which to base them. Therefore, the first four or five weekly programming assignments are always completed using random pairings.

A further potential weakness in the dataset is the fact that the two sections that employed random pairings were also exceptional in another respect: in both cases, these sections were taught by instructors who had not taught the course before. Therefore, inferior performance by students in random-pair sections might be due to the relative unfamiliarity of the instructors with the course material. To investigate whether this is indeed a serious problem, we analyzed the outcomes of the *first* programming exam. This exam is given in week 7 of the semester, so students in the ability-pair sections have worked on only one or two assignments in by-ability pairs. If the effect of first-time instructors dominated the effect of random pairings, we would expect a difference between the random-pair sections and ability-pair sections on *both* programming exams. However, the 25th-percentile difference on the first exam is not significant (p-value 0.41), which suggests the instructor effect is not dominant and adds credibility to the previous analysis.

4. DISCUSSION

The evidence presented suggests that pairing students of similar ability may improve the performance of less able students on individual programming tasks as compared to random pairings. Why might this be the case? We can envision several reasons.

First, because our ability metric of performance in the course to date includes some direct measures of the tasks to be completed in pairs (actual programming), it seems likely that students who are paired together will perceive

each other to be of similar ability, and thus are likely to be compatible. Each of the studies cited earlier that did not observe a correlation between actual ability and pair compatibility used less specific measures of actual ability such as GPA, SAT/GRE scores and midterm exam scores. In fact, one of these studies alludes to exactly this, saying that while pair compatibility did not correlate with metrics such as GPA, SAT/GRE and midterm scores, more specific metrics such as computer science GPA and total GPA are predictors of perceived ability [17]. Thus, we suspect that our mechanism for pairing by ability is in fact forming more compatible pairs than pairing randomly.

Second, and of particular relevance to the performance of the weaker students in the class, are the implications of working with a partner of similar ability. One might be tempted to pair a weaker student with a stronger student so that the weaker student can learn from the stronger. But in our experience, and in that of others [4, 10], the stronger student instead takes over, and either does the work or simply gives direction. Thus, in random pairings that contain a stronger and weaker student, the weaker student will not actually experience the process of resolving whatever difficulties arise — instead simply observing how the stronger student handles the problem. Even worse, when the stronger student is the driver in the pair, the weaker student may not even be exposed to problems that he/she would face if working individually. When two students of similar ability work on such a problem together, they are more likely to resolve it in a collaborative way that seems likely to result in deeper understanding and a better learning outcome for both students.

5. CONCLUSIONS

If pairing by ability can improve outcomes for the students who have demonstrated the least aptitude for the course material, further investigation certainly seems warranted, particularly since this approach requires no extra class time and little administrative overhead, and appears to have no harmful effects on higher-ability students [1]. A more tightly controlled study comparing the performance of students in sections that are paired randomly to those that are paired by ability would not be difficult to perform. If such an investigation were to be conducted it should also include self-pairing, as at least one study found that when students are allowed to self-pair they tend to choose partners that have a similar ability level [12]. Thus, perhaps any benefits of ability pairing could be obtained without the even minimal administrative overhead of assigning pairs.

6. ACKNOWLEDGMENTS

Partial funding for this research was provided by the National Science Foundation under a CCLI-A&I grant (DUE-0511264). We would also like to thank Dr. Stephen Edwards of Virginia Tech for supporting our use of WebCAT, which provided quantitative data on student program quality and testing.

7. REFERENCES

[1] G. Braught, L. M. Eby, and T. Wahls. The effects of pair-programming on individual programming skill. In *SIGCSE*, pages 200–204, 2008.

[2] G. Braught, T. Wahls, and M. Eby. The case for pair programming in the computer science classroom. *Submitted to Trans. Computing Education*, 2009.

[3] Carver, Henderson, He, Hodges, and Reese. Increased retention of early computer science and software engineering students using pair programming. *CSEET*, pages 115–122, 2007.

[4] E. A. Chaparro, A. Yuksel, P. Romero, and S. Bryant. Factors affecting the perceived effectiveness of pair programming in higher education. In P. Romero, J. Good, S. Bryant, and E. A. Chaparro, editors, *Psychology of Programming Interest Group 17th Workshop*, pages 5–18, 2005.

[5] B. Hanks. Student attitudes toward pair programming. In *ITiCSE*, pages 113–117, 2006.

[6] N. Katira, L. Williams, and J. Osborne. Towards increasing the compatibility of student pair programmers. In *ICSE*, pages 625–626, 2005.

[7] N. Katira, L. Williams, E. Wiebe, C. Miller, S. Balik, and E. Gehringer. On understanding compatibility of student pair programmers. In *SIGCSE*, pages 7–11, 2004.

[8] C. McDowell, L. Werner, H. E. Bullock, and J. Fernald. The impact of pair programming on student performance, perception and persistence. In *ICSE*, pages 602–607, 2003.

[9] C. McDowell, L. Werner, H. E. Bullock, and J. Fernald. Pair programming improves student retention, confidence, and program quality. *Commun. ACM*, 49(8):90–95, 2006.

[10] G. Melnik and F. Maurer. Perceptions of agile practices: A student survey. In *Proc. Extreme Programming and Agile Methods*, pages 241-250, 2002.

[11] E. Mendes, L. Al-Fakhri, and A. Luxton-Reilly. A replicated experiment of pair-programming in a 2nd-year software development and design computer science course. *SIGCSE Bull.*, 38(3):108–112, 2006.

[12] R. Nicolescu and R. Plummer. A pair programming experiment in a large computer course. *Romanian J. Inf. Sci. and Tech*, 6(1-2):199–216, 2003.

[13] P. Sfetsos, I. Stamelos, L. Angelis, and I. Deligiannis. An experimental investigation of personality types impact on pair effectiveness in pair programming. *Empirical Softw. Eng.*, 14(2):187–226, 2009.

[14] L. Thomas, M. Ratcliffe, and A. Robertson. Code warriors and code-a-phobes: a study in attitude and pair programming. In *SIGCSE*, pages 363–367, 2003.

[15] Web-cat research server front page. http://web-cat.cs.vt.edu/, 2009.

[16] L. L. Werner, B. Hanks, and C. McDowell. Pair-programming helps female computer science students. *J. Educ. Resour. Comput.*, 4(1):4, 2004.

[17] L. Williams, L. Layman, J. Osborne, and N. Katira. Examining the compatibility of student pair programmers. In *Proc. AGILE*, pages 411–420, 2006.

[18] L. Williams, L. Layman, K. M. Slaten, S. B. Berenson, and C. Seaman. On the impact of a collaborative pedagogy on african american millennial students in software engineering. In *ICSE*, pages 677–687, 2007.