Lessons Learned from Seven Years of Pair Programming at North Carolina State University

Laurie Williams
North Carolina State University
Department of Computer Science
williams@csc.ncsu.edu

Abstract

A recent survey conducted on the SIGCSE mailing list indicated that up to 80% of CS1, CS2, and data structures instructors allow students to collaborate. The use of collaboration increases as students advance through the computer science curriculum. Some computer science educators use pair programming as the model for their student collaboration, sometimes with mixed results. At North Carolina State University, over a thousand students have pair programmed in CS1, undergraduate software engineering, and graduate level courses over the last seven years. This paper provides a summary of the lessons we have learned through experience and through extensive research over this time period.

1. Introduction

In January 2007, 148 SIGCSE educators responded to an online survey regarding the practice and perceptions of collaboration in the classroom [6]. The results of the survey indicate that up to 80% of the instructors of CS1, CS2, and data structures courses allow students to collaborate. The use of collaboration generally increases as students advance through the computer science curriculum into courses such as software engineering or computer graphics. Some computer science educators use pair programming [5] as the model for their student collaboration.

Pair programming is a style of programming in which *two* programmers work side-by-side at *one* computer, continuously collaborating on the same design, algorithm, code, or test. One of the pair, called the *driver*, types at the computer or writes down a design. The other partner, called the *navigator*, has many jobs. One is to observe the work of the driver, looking for defects. The navigator also has a more objective point of view and is the strategic, long-range thinker. Together, the driver and the navigator continuously brainstorm a solution. Periodically, the programmers switch roles between the driver and the navigator.

Research has suggested many pedagogical benefits to pair programming, including that pair programming creates an environment conducive to more advanced, active learning and collaboration, leading to students being less frustrated, more confident, and more interested in information technology [1, 2]. But, as will be discussed, successfully transitioning to pair programming in the classroom is dependent on several factors. At North Carolina State University (NCSU), we have used pair programming extensively in our CS1, undergraduate software engineering, and several graduate level courses over the last seven years, involving more than a thousand students. We have learned many lessons and have adapted our policies and practices to be more successful with the collaborative pedagogy of pair programming, including balancing a mix of pair programming and solo programming in each class. This paper summarizes the lessons we have learned in hopes of enabling other educators to be as successful as possible with pair programming.

2. The Positive and Negatives Aspects of Student Pair Programming

As with most pedagogical techniques, pair programming has positive and negative impacts on both the student and the instructor. Additionally, pair programming appears to increase retention [8], particularly among women, because it reduces some negative aspects of our traditional pedagogy that focused on solo programming, especially in the early years of the curriculum. Through pair programming, students are able to meet the other students in their classes and laboratories, thereby creating a more friendly and supportive environment. Students in the Millennial generation desire to work collaboratively [4]. Also, pair programming helps to prepare students for the collaboration, teamwork, and communication skills required in industry.

Pair programming benefits the teaching staff as well. These benefits include less grading due to half the number of assignment submissions and less cheating because of the innate support structure of pairs. Also, a pair of students can often figure out the low-level technical or procedural questions that oftentimes burden the instructor's office hours and email inbox. Students enjoy the collaborative environment, and therefore are more positive and happier about the class. Finally, there are fewer "problem students" to deal with because the peer pressure involved in pair programming most often results with all students turning in a decent assignment. Some students become more concerned about jeopardizing their partner's grade and end up working harder on the assignments, often getting started earlier than if they worked alone.

Alas, there are some costs to implementing pair programming. Most of the costs for educators are outlined in the "lessons learned" of the next section. For students, there are two major costs that persist without apparent recourse. First, a small segment of students will always desire to work alone. Most often, these are the top students who do not want to be bothered with being "slowed down by" another student. Another problem for students is the need to coordinate schedules with another student when pair programming is required outside of a classroom or laboratory setting.

3. Lessons Learned

In this section, we enumerate the lessons we have learned in hopes of enabling other educators to be as successful as possible with pair programming:

Closed Laboratory. Based on our experience, undergraduate students should have extensive experience (probably two full courses) with pair programming in a closed laboratory setting before they can realistically pair on their own outside of class. The watchful eye of a trained instructor or teaching assistant is essential for making sure that students are properly assuming the roles of driver and navigator, switching roles periodically, and that both students are engaged. Additionally, inexperienced students seem to have more difficulty meeting with their partners outside of class. Students with more experience pair programming are better able to manage meeting outside of class, but these students still seem to benefit from "bonding" with their partner by working on a joint project in a structured lab setting. We do not require any pair programming outside of the closed lab for our CS1 class.

Attendance and Lateness Policy. A strict attendance in lecture and laboratory has always been required for our CS1 class but not the others. For several reasons, we have evolved to a relatively strict attendance policy for the undergraduate software engineering class as well. First, a student who does not attend lecture impacts his or her ability to work effectively on and to contribute fairly to a paired project. Secondly, we are more likely to identify earlier those students who are no longer participating in the class but have not dropped it yet. Additionally, we were having a problem with students arriving late for laboratory, wondering what to do with the student anxiously awaiting the arrival of a partner. We recommend that after a specified period of time (e.g. 10 minutes), a student is reassigned to a different pair if their partner does not arrive. A student who arrives late must work alone with a penalty on the lab assignment.

Peer Evaluation. Some students might attempt to escape an assignment without doing the work and/or learning the lessons the assignment is designed to teach. An important mechanism for providing the motivation to participate is a formal peer evaluation process that requires students to provide feedback on the participation of their partner. At the end of each paired homework assignment, we require students to evaluate their partners using the online PairEval[1] system. Based upon the peer rating system by Kaufman et al. [3], the students are asked to choose one of the 9 key words in the bullets below (short descriptions are provided to the students as well) to describe the contribution of his or her partner. We found that asking the students to choose a word to describe the contributions of their partner was more effective than asking for a numerical rating. Previously, the students all tended to give their partner a high numerical score, and now there is a wider range of response. The students choose among the following ratings:

1. **Excellent.** Consistently went above and beyond—tutored teammates, carried more than his/her fair share of the load
2. **Very good.** Consistently did what he/she was supposed to do, very well prepared and cooperative
3. **Satisfactory.** Usually did what he/she was supposed to do, acceptably prepared and cooperative
4. **Ordinary.** Often did what he/she was supposed to do, minimally prepared and cooperative
5. **Marginal.** Sometimes failed to show up or complete assignments, rarely prepared
6. **Deficient.** Often failed to show up or complete assignments, rarely prepared

[1] Pair Eval can be freely downloaded for use at http://agile.csc.ncsu.edu/pairlearning/paireval.php.

7. **Unsatisfactory.** Consistently failed to show up or complete assignments, unprepared
8. **Superficial.** Practically no participation
9. **No show.** No participation at all

Students can also input into the system the rationale behind their rating. If a student gives their partner a low overall rating (e.g. "Marginal" or below), the partner is flagged and the teaching staff can review the evaluation more carefully and perhaps contact the student. If, after such a discussion, the instructor determines that a student made little or no effort on a partnered assignment, the student will have his or her grade reduced accordingly and the partner's grade will be correspondingly increased. We use a strict policy whereby a student's score can be multiplied by their contribution (e.g. if they did 50% of what they were supposed to, they get 50% of the score). When pairing students for homework assignments, prompt attention and adequate consequences are essential to bring potential "freeloaders" back into a contributing role. After the instructor handles a few of these instances, an environment of participation is created in the classroom and instances of freeloading become rare. We recognize that not all freeloaders are identified via this peer evaluation process because sometimes students are reluctant to report their partners.

Grading. We want to ensure that individual students are learning the course material and not relying solely on their partners. In CS1, where students are establishing foundational knowledge required for the rest of the curriculum, the weekly collaborative lab assignments count for 10% of the overall grade. The other 90% of the overall grade is assessed through individual activities, such as exams, written homework, and other programming assignments. In other classes, where larger projects are done collaboratively, we have a policy that the students must have a passing grade in the individual portions of the class in order to pass the class.

Choosing the pairs. We assign the pairs rather than allow them to choose their partners. Through the Pair Eval system, we monitor the compatibility of our pairs in addition to their contribution. We have found that less than 9% of pairs report compatibility problems [7]. Students consistently express a desire to work with a partner of equal or better skill level relative to themselves. We have examined a variety of factors to determine if we can proactively assign effective student pairs. We have examined the use of SAT scores, grade point average (GPA), computer science GPA, Myers Briggs compatibility components, Felder-Silverman learning components, a computer science self esteem index, work ethic, and time management factors. We have found the only factors that can be used to proactively managed pairs are midterm scores (only available halfway through the semester), pairing a Myers Briggs sensor with an intuitor, or pairing students of similar work ethic [7]. We determine their work ethic by asking the students to rate themselves on a scale from 1 to 9 on the following question in a pre-class survey administered via Pair Eval:

In your classes, do you work hard enough to:
1: Just barely get by
9: Get the best grade you possibly can.

Teaching assistant training. The teaching assistants (TA) need coaching and training on how to manage a pair programming lab. In the laboratory, they must look for dysfunctional pairs who are not working well together. The TAs must approach the pair and ask if they need help, and point them in the right direction for working together via driver/navigator roles. The TAs must ensure that the students switch roles periodically. In solo programming labs, the TAs spend all of their time answering questions of the students. In a paired lab, the role of the TA changes, to some degree, from technical assistant to proactive monitor. For the most part, the multitude of technical questions of the TAs in solo labs is reduced because pairs can usually figure out most aspects of an assignment together. Questions from pairs tend to be focused more on learning objectives and concepts rather than technical hang-ups, and may require more time per question (though this is time well-spent). With the remaining time, the TA needs to proactively visit the pairs, asking how they are doing, and ensuring that they are working together effectively.

Student training. Likewise, students need training in pair programming. The instructor cannot assume that the students will figure out what to do in pairs; they may feel the idea is to divide the work into two parts, each student doing half. The students should be made aware that they are to work together at one computer in driver and navigator roles, they need to switch roles, they both to be active participants at all times, and so forth. Resources for educating students about pair programming, including a 15 minute video, can be found at http://agile.csc.ncsu.edu/pairlearning/.

Prompt attention to problems. The most common problem with pairing is non-participation on the part of one student. Students must understand that problems with their partner must be surfaced early to give the instructor a chance to correct the situation. The instructor must understand the non-participatory student's perspective as well, and/or ask the student if he or she intends to continue with the class. If the instructor determines that a student is not going to contribute fairly to an assignment, then the partner will be given reparations on the assignment, such as the

option to complete a subset of the assignment individually. When students report problems at the last minute, they are not given these same reparations.

Pair rotation. We assign students new partners at least three or four times per semester. Periodically assigning new partners is beneficial for the students because they have the opportunity to meet more of their peers. Also, students will be less likely to be intolerant of their partner if they know their "relationship" only lasts a week or two. Rotating pairs is beneficial for the teaching staff because obtaining multiple forms of peer evaluation on each student provides a more accurate picture of the contributions of the student. Additionally, the regular pair rotation allows dysfunctional pairs to separate without overt action on the part of the instructor.

Lab setup. At NCSU, the software engineering laboratory has an ideal pair programming setup. All computers in the lab have two monitors, two mice, and two keyboards and the room is arranged as shown in Figure 1 to enable pair-to-pair communication. A more traditional setup is fine if two people can sit comfortable next to each other where both can see the display (generally a six-foot table per pair). The litmus test is that the driver and navigator should not need to switch chairs when they switch roles. If they need to switch chairs so the driver can be better positioned, then the set up is not ideal for pair collaboration.

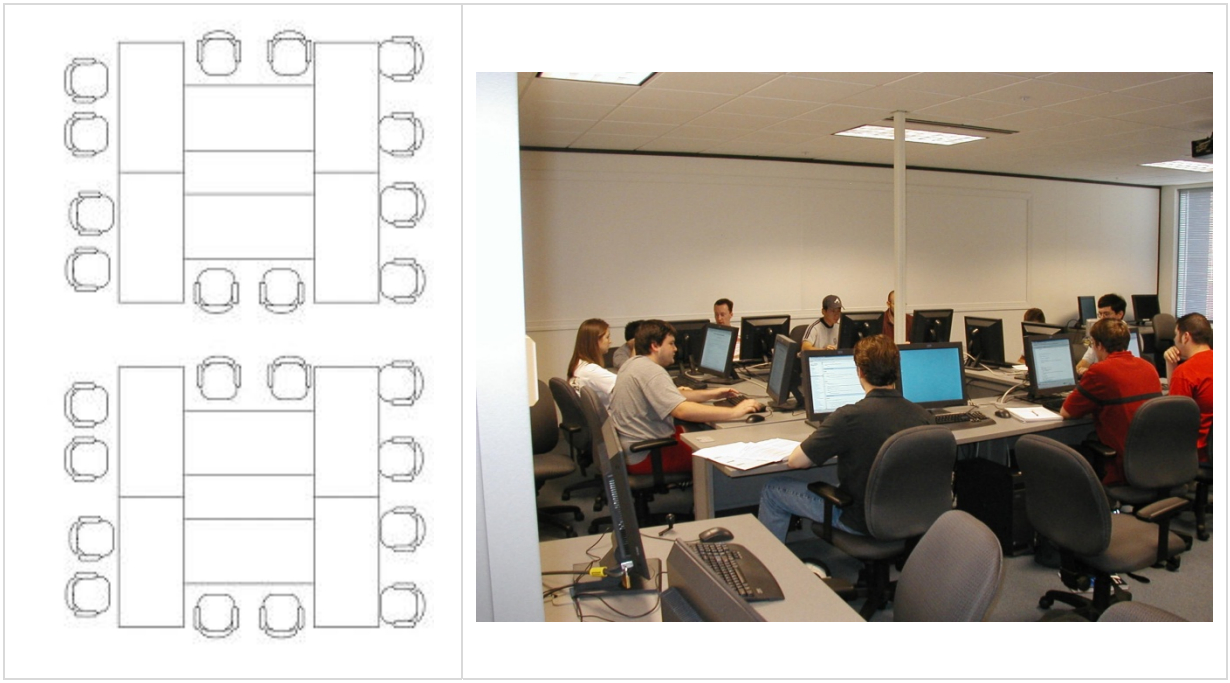

Figure 1. Software engineering lab

4. Summary

Based upon our experiences with over a thousand pair programming students in the last seven years, we believe strongly in the benefits of a balanced use of this pedagogical technique in computer science classes, beginning with the very first class. We have learned many lessons that helped us to be more effective with the technique over time. We hope by sharing these lessons, other educators will allow their students to pair program and that the transition to pair programming is as smooth and beneficial as possible for student and instructor alike.

5. Acknowledgements

I would like to acknowledge the instructors and teaching assistants at North Carolina State University who have helped to institute pair programming: Carol Miller, Suzanne Balik, Ed Gehringer, Matthais Stallman, Lucas Layman, Mark Sherriff, Sarah Smith Heckman, Kristy Boyer and many others. Additional thanks to Lucas, Mark, and Sarah for providing feedback on earlier drafts of this paper. This material is based upon the work supported by the National Science Foundation under Grants CCLI 29728000, ITWF 00305917, and BPC 0540523. Any opinions,

findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

6. References

[1] S. B. Berenson, K. M. Slaten, L. Williams, and C.-w. Ho, "Voices of Women in a Software Engineering Course: Reflections on Collaboration " *ACM Journal on Educational Resources in Computing*, vol. 4, no. 1, March 2004.

[2] S. B. Berenson, L. Williams, and K. M. Slaten, "Using Pair Programming and Agile Development Methods in a University Software Engineering Course to Develop a Model of Social Interactions," in *Crossing Cultures, Changing Lives Conference*, Oxford, UK, 2005, p. to appear.

[3] D. B. Kaufman, R. M. Felder, and H. Fuller, "Peer Ratings in Cooperative Learning Teams," in *American Society for Engineering Education*, Charlotte, NC, 1999.

[4] D. Oblinger, "Boomers, Gen-Xers, and Millennials: Understanding the New Students," *Educause Review*, vol. 38, no. 4, pp. 37-47, July/August 2003.

[5] L. Williams and R. Kessler, *Pair Programming Illuminated*. Reading, Massachusetts: Addison Wesley, 2003.

[6] L. Williams and L. Layman, "Lab Partners: If They're Good Enough for the Natural Sciences, Why Aren't They Good Enough for Us?," in *Conference on Software Engineering Education and Training*, Dublin, Ireland, 2007, pp. 72-82.

[7] L. Williams, L. Layman, J. Osborne, and N. Katira, "Examining the Compatibility of Student Pair Programmers," in *Agile 2006*, Minneapolis, MN, 2006, pp. 411-420.

[8] L. Williams, C. McDowell, N. Nagappan, J. Fernald, and L. Werner, "Building Pair Programming Knowledge Through a Family of Experiments," in *International Symposium on Empirical Software Engineering (ISESE) 2003*, Rome, Italy, 2003, pp. 143-152.