Guidelines for the Use of Pair Programming in a Freshman Programming Class

Jennifer Bevan, Linda Werner, Charlie McDowell
Department of Computer Science
University of California at Santa Cruz
{jbevan,linda,charlie}@soe.ucsc.edu

Abstract

Undergraduate freshman programming classes are conventionally organized such that individual students complete a set of concept-specific and unrelated programming assignments. This structure does not prepare students for future collaborative efforts or for the future use of software engineering practices. The addition of pair programming into a freshman programming class at the University of California at Santa Cruz (UCSC) showed similar benefits to similar studies on upper-division software classes[1,2], and is expected to show an improvement in students' willingness and ability to participate in complex, collaborative software engineering assignments in later classes. This paper describes the implementation of the pair programming experiment at UCSC, discusses some of the issues that compromised the effectiveness of certain pairs, and provides implementation guidelines for avoiding such issues in other classes.

1. Introduction

Conventional undergraduate freshman programming classes do not prepare students for either collaborative efforts or software engineering practices. These classes generally require that individual students complete a set of concept-specific assignments that do not build upon each other. The use of such assignments does not teach students the importance of good design and maintenance practices; students do not have to cope with problems within the previous week's assignment[3]. This problem is difficult to overcome because freshman classes are mainly comprised of students who have little or no programming experience. The necessary small and concept-specific assignments do not easily support the overhead of a software engineering methodology, and most students do not have the requisite background to write even small unit tests. On the other hand, the issue of preparedness for later collaborative efforts can be addressed by the addition of pair programming into the class structure.

Pair programming in freshman classes does not inherently suffer from unique hardships uncommon to pair programming in other university and industry settings. The most common issues that can detract from the pair programming experience are grounded either in attitude conflicts or scheduling conflicts[2,4]. Attitude conflicts can be found in both industry and university settings, and can be addressed through management buy-in and professional conduct[5]. Scheduling conflicts are more strongly felt in the university: industry can encourage compatible employee schedules while universities do not restrict course selection beyond a limit in the total number of units taken per term[6].

Pair programming offers a method of breaking the pattern of conditioning students to work alone. Although researchers have agreed that student programmers are conditioned to equate communication and sharing with cheating[1,4], the conditioning is widespread in more than just the computer science community. Throughout junior high and high school, students are conditioned that working together is the same as cheating. While an assessment of individual competency is necessary for professional or academic certification, many teachers and administrators view collaboration as an unwieldy grading problem, and prefer to avoid it entirely. In recent years, however, encouraging studies on the effectiveness of pair programming in junior and senior level undergraduate classes have been accomplished[1,2]. By introducing pair programming into undergraduate freshman classes, this pattern can be broken earlier in the programmers' development.

An ongoing NSF-funded study, conducted at the University of California at Santa Cruz (UCSC), restructured the freshman programming course to use pair programming as a means of determining both its effectiveness as a teaching technique and its effect on student retention within the computer science program. The preliminary results, which contrast the performance of successful pairs to individual students, make a significant case for pair programming[7]. This paper describes the implementation of the restructured freshman programming class and discusses the issues that caused problems among the students. A set of implementation guidelines is then provided for instructors interested in adopting pair programming within their own classrooms.

2. Background

When pair programming, two developers alternate between the roles of *driver* and *reviewer*. The driver issues all keyboard and mouse input within the development environment. The reviewer observes the input, mentally compares the actual functionality of the new code with the expected functionality, and maintains active communication with the driver regarding the correctness and appropriateness of the code.

Pair programming inherently incorporates basic design and review phases in the development process. The programmers must communicate about possible approaches in order to discuss appropriateness during development. Both members review the other's input, a process that increases the developers' ability to perform effective reviews.

The first paper (McDowell et. al.) from our study gives a more thorough description of the evolution and practice of pair programming[7].

3. Implementation details

During the 2000-2001 academic year, four lecture sections of the introductory programming class took part in our study. The Fall and Winter quarter classes all used pair programming while the Spring class did not. The Fall and Spring classes were taught by the same instructor. Each class completed 9 assignments in 10 weeks. Four of the assignments were practice problems that did not directly contribute to a student's grade. The *n*th-week assignments for each class generally focused on the same programming topic, with some variation in difficulty between instructors. An entrance

and an exit survey were performed in each class; previous programming experience and strength in mathematics were among the data solicited.

Students in the Spring class, which did not use pair programming, were required to complete time logs for each assignment, indicating total work time, satisfaction with their solution, and satisfaction with the experience of solving the problem. The students in the paired classes were required to complete similar time logs that also indicated their satisfaction with the experience of working with a partner and the times spent driving, reviewing, and working alone.

For the Fall and Winter classes, the pair programming requirement was that no less than 75% of the total time spent by a student on an assignment was to be physically spent with their partner. Students were to alternate between driving and reviewing at intervals of no more than 1 hour. These students were also given copies of Williams and Kessler's "kindergarten" paper[4], and links to other pair programming papers were placed on those class web sites. Students were re-paired only if the pair had insurmountable scheduling difficulties or if one partner repeatedly did not come to scheduled meetings. If requested, students were allowed to work alone, but they were required to fill out the same time logs as the other students.

Pairing was based on student requests: each student turned in a list of three other students in the order of pairing preference. In the Fall quarter, the pairing scheme required that paired students be enrolled in the same section. This was not the case with the Winter quarter classes; in those classes the students' preferences were turned into a weighed graph, where edge weights were a linear function of two people's preference to work with each other. The edges were ordered by weight and the pairs selected from this ordered list. Students not selected by this process were randomly paired with other such students. The pairing process resulted in nearly the same number of women paired with other women as were paired with men, which was important for our study. Programming experience and grade point average were not taken into account during pairing.

4. Sources of difficulties

Three of the four classes in this study were taught by different instructors, with the help of eight different graduate student teaching assistants (TAs). In each case, the instructor gave the lectures while the TAs oversaw the computer lab sections. Variances in the implementation of pair programming between the different classes, and the effects of these variances on the stability of the student pairs, were obtained through TA interviews, optional comments from the required student logs, and direct student email.

Several pair-related issues were discovered to be common amongst all of the paired classes in the study, each of which related to how well the students worked together. The two most common sources of intra-pair stress were a significant disparity between the experience levels of the students and difficulties with scheduling or reliability. The students' methods of dealing with these stresses sometimes resulted in unintentional cheating, due to confusion regarding the relative importances of required pairing and honesty.

4.1. Effects of class structure

Most of the class-level differences resulted from variances in the enforcement and reinforcement of the pair programming requirements. The instructors did not uniformly emphasize what was expected of the paired students during lectures, which resulted in several cases of misunderstandings during the first several weeks. For example, some pairs were found to be partitioning the problems and working on those partitions separately, instead of using pair programming. Others were e-mailing the code back and forth, incrementally completing the assignments separately. The extent to which TAs could identify such behavior was tied to the instructor's section policy: mandatory lab sections increased the TA's ability to verify that the paired students were following the pair programming requirements. Given that paired students were not necessarily in the same lab section, the TAs were not instructed to restrict access to the computers if both partners were not present. Therefore, partitioning or alternating development strategies were not always actively discouraged.

4.2. Student scheduling and reliability problems

Students were encouraged to alert the researchers conducting the study if their partner was repeatedly unreliable or if both partners had insurmountable scheduling conflicts. Unfortunately, some pairs tried to overcome scheduling conflicts for several weeks before reporting the problem, which made re-pairing much more difficult. An average of less than 5% of the pairs in each class reported such issues and were re-paired if possible.

Optional comments on the logs led to several discoveries about scheduling conflicts and unreliable partners that were not directly reported. One of the more surprising discoveries was the willingness of students to submit an assignment with both partners' names attached, even if one partner had not contributed at all. An investigation revealed that the students in question believed that it was more important that they appear to be following the pairing requirements than it was to be honest about the division of labor.

4.3. Differing levels of experience between partners

The introductory programming class is a required course in the computer science program at UCSC, with a pre-requisite of pre-calculus but no prior programming. This led to a wide variance in experience levels among students. The portion of the classes that did not self-pair were randomly paired without knowledge of relative experience levels. While our study indicates that both members of a functional pair benefitted from pair programming, the probability of the pair being functional was lessened by a significant disparity in experience levels. The more experienced students were frequently unwilling to explain the relevant concepts to the other, or to wait for the other to understand the material. Several students described having a less adept partner as a “waste of their time”, and would simply write the entire program alone and submit it as a combined effort. Others stated that because their programming style suits them, and because working with another would force them to change their style, they would not participate in a paired process. Just under 2% of the pairs were discovered to have this type of problem to such a degree that they were eventually re-paired.

4.4. Student understanding and buy-in

The submitted student logs identified several pairs that were not conforming to the pair programming guidelines due to a lack of understanding or commitment. Some pairs required personal, in-depth descriptions of what pair programming was and what was to be accomplished by the members of a pair. Several students equated "working alone" with "driving", and logged double their actual time spent on the assignment. Several pairs showed an unacceptable imbalance of driving and reviewing times. In one such case, a student was extremely uncomfortable at the keyboard. In others, one partner admitted authorship of most or all of the code submitted thus far. Some students thought that they were in compliance but were not; others recognized that they were not following the pair programming guidelines but did not care. None of these issues forced a pair to be split up, but the efficiency of the pair was reduced until the problem was worked out. This type of problem can be difficult to detect without the active involvement of the TAs during the computer labs; the student logs are not sufficient to identify every occurrence of this behavior.

5. Implementation guidelines

Our study has indicated that pair programming improves a student's performance on programming assignments in an introductory programming class[7]. Even so, the structure of the class can fail to encourage and enforce the pair programming environment. The following guidelines address the implementation issues described above, and should be used as a framework by instructors interested in adopting pair programming.

1. *Pair within sections*. Pairing within sections helps TAs identify students who are not conforming to the pair programming guidelines because the missing partner is known to be in the same section. During the scheduled lab sections in the first week of class, the TA should oversee introductory activities that allow students to find a compatible partner within their section. These activities should candidly address scheduling compatability and student concerns about working with a partner. Discussion can be facilitated by the application and analysis of personality profiling tests, given the caveat that the results should not determine the pairings. Potential pairs can perform a "test run" of the partnership by jointly solving small jigsaw puzzles or other similar games that are not independently solvable. Although odd-numbered enrollments are impossible to avoid, some students will have a more flexible schedule than others, and could switch sections if necessary. This extra effort during the first week of sections will help to provide the "jelling" time necessary for successful pairing[8].
2. *Pair (somewhat) by skill level*. Although skill level is not as crucial a pairing parameter as co-enrollment in a given section, pairing with regard for skill level can avoid some compatibility problems. Freshman undergraduates do not necessarily believe that the axiom "you always learn more when you teach somebody else" applies to them. The resulting impatience between partners is detrimental to the effectiveness of a pair. This guideline is targeted at classes that introduce pairing to students. Students who state that they are unwilling to work with a less experienced student can be paired at a similar skill level, thereby increasing the stability of that

pair. Once students are comfortable with the idea of pairing, they are more likely to be more open-minded when working with partners that are not as well-matched[2,8].

3. *Make sections mandatory*. Even if partners are enrolled in the same lab section, partner reliability can still be a problem. Mandatory lab sections ensure that some minimal amount of time is spent working on the assignment together. The instructor should define a set of acceptable excuses, such as illness or having already completed the assignment, that will excuse a pair from the section. If possible, both partners should inform the section TA if they wish to invoke one of these excuses.
4. *Assignments as a function of section time*. First-year programming classes generally focus on a few basic concepts per assignment. Some assignments, however, can require a lot of other work that extends the expected time-to-completion far beyond the total time of the scheduled lab sections. If instructors create such assignments, paired students are more likely to encounter scheduling difficulties that make it impossible to conform to the pair programming guidelines. Instructors need to tailor the expected length of the assignments such that a reasonable percentage of the class can finish within the scheduled section time. This practice improves the ability of the students to maintain the pair programming process and sacrifices no coverage of concepts.
5. *Institute a coding standard*. Experienced programmers generally have both a personal coding style and the ability to conform to any required coding standard. Alternately, less-experienced programmers tend to view their personal coding style as "right", and anything different as "wrong". Within a pair programming class, an instructor-selected coding standard can smooth out such differences between partners because neither individual can dominate the coding style. The resultant decrease in friction increases the effectiveness of the pair.
6. *Create a pairing-oriented culture*. Classes with pair programming issues integrated into their structure can improve the collaborative process without increasing the level of oversight required by the TAs or professor. For example, scheduling conflicts will happen, and therefore assignment submission and grading policies need to accommodate this. Suppose that a pair was not able to finish debugging a program together. An official method of separately turning in individually debugged program will reduce the chance that one partner will finish the program and submit it using both partners' names. Forcing a student to depend upon an unreliable partner in order to follow both grading requirements and pairing guidelines can create a conflict between honesty and academic compliance. Some students have grown up in an academic tradition that emphasizes academic compliance above all else. In order to counteract such training, the TAs or the instructor should, on a weekly basis, discuss some of the ideals behind pair programming with the students. Discussion on how the implementation of pair programming in the class differs from those ideals should be encouraged. The open discussion of issues such as collective ownership of code, maintenance goals, and industry expectation does not have to take up a lot of class time and can foster a culture within the class that supports the goals of pair programming.

   Ideally, the extent to which students attempt to follow the pair programming guidelines should affect their final grades in the class. Unfortunately, determining where pairs went wrong would require more time than instructors and TAs usually have. The existence of a class culture that encourages pair programming will help students realize when they are drifting away from the pair programming guidelines, and we hope will help them identify and self-correct the problem.

6. Conclusions

Pair programming surveys have shown that experienced pairs will frequently work alone on rote or simplistic implementation details[4,8]. For most freshman undergraduates, however, there are no rote details: everything is new and difficult. Within a paired curriculum, the pair-learning, pair-relaying, and pair-think behaviors discussed by Williams "create a unique educational capability, whereby the pairs are endlessly learning from each other" that can be beneficial for these freshmen[2]. We emphasize, however, that the implementation of pairing within the freshman classroom requires careful attention to detail; behaviors that are not common in industry or even upper-division classes can undermine the stability of a given pair. In this setting, the most critical aspect of creating an effective pair programming implementation is to minimize the potential scheduling conflicts between partners. The additional support of a culture that emphasizes cooperation, mutual respect, and shared responsibility paves the way for partners to work out attitude-based problems.

Even though the percentage of pairs that were reassigned due to the problems described was quite low, an effort should still be made to reduce this percentage further. The guidelines suggested in this paper can be used by other instructors as a means of more effectively structuring their freshman classes to prepare their students for future collaborative work. As more studies are completed, these guidelines should, of course, be refined.

7. Future Work

The ongoing project at UCSC will monitor the students from the studied freshman classes for the remainder of their undergraduate career. Data such as the types of classes chosen, performance in such classes, and attitudes about computer science and software engineering issues will be collected. We hope to determine the effects of pair programming on retention percentages within the computer science field, personal software processes, and long-term interest in software engineering.

8. Acknowledgements

Thanks to Dr. Heather Bullock, Dr. Julian Fernald, Wendy R. Williams, M.S, and Tristan Thomte, from the Psychology Department at UCSC, for assistance in creating the surveys, data collection, and analysis, to Dr. Alex Pang and Dr. Scott Brandt of the Computer Science Department at UCSC for participating in the pair programming study, to the anonymous reviewers for CSEET 2002, and to the National Science Foundation for the project funding (NSF EIA-0089989, "Retaining women in computer science: Impact of pair programming"). 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.

9. References

[1] L. Williams, "But, isn't that cheating? [collaborative programming]," *FIE'99 Frontiers in Education. 29th Annual Frontiers in Education Conference*, vol. 2, pp. 12B9/26-27, Nov. 1999.

[2] L. Williams, "Integrating pair programming into a software development process," *Proceedings 14th Conference on Software Engineering Education and Training*, pp. 27-36, Feb. 2001.

[3] R. Kessler and L. Williams, "If this is what it's really like, maybe I better major in English: integrating realism into a sophomore software engineering course," *FIE'99 Frontiers in Education. 29th Annual Frontiers in Education Conference.*, vol. 1, pp. 12A4/12-16, Nov. 1999.

[4] L. Williams and R. Kessler, "All I really need to know about pair programming I learned in kindergarten," *Communications of the ACM*, vol. 43, pp. 108-114, May 2000.

[5] J. Haungs, "Pair programming on the C3 project," *Computer*, vol. 34, pp. 118-119, Feb. 2001.

[6] J. Kivi, D. Haydon, J. Hayes, R. Schneider, and G. Succi, "Extreme programming: a university team design experience," *2000 Canadian Conference on Electrical and Computer Engineering.*, vol. 2, pp. 816-820, Mar. 2000.

[7] C. McDowell, H. Bullock, J. Fernald, and L. Werner, "The effects of pair-programming on performance in an introductory programming course," *Accepted by SIGCSE 2002*.

[8] L. Williams, R. Kessler, W. Cunningham, and R. Jeffries, "Strengthening the case for pair programming," *IEEE Software*, vol. 17, pp. 19-25, July 2000.