Utilizing Pair Programming Techniques in a Web Development Course

Troy Harding
Kansas State University at Salina

Abstract

Recent research has demonstrated pair programming to be an effective model of collaborative learning for millennial students. However, much of the research describes the impact of pair programming in first-year programming courses. This paper highlights experiences using pair programming in an advanced web development programming course. Most of the students in the course were at the junior or senior level and had not been formally introduced to pair programming prior to taking the course. Qualitative data were collected through the use of written reports and surveys. The author discusses what was learned about the impact on students' attitudes, learning and quality of work. Challenges are also described, as well as recommendations for enhancements.

Introduction

Computer programming has traditionally been a solo activity. Even in teams of software developers, the planning of the project may be done together, but the actually coding is typically done individually. In recent years the growing popularity of the extreme programming software development approach has brought attention to pair programming.[1,2] Pair programming is where two programmers work together at one computer. They continuously collaborate on designing, coding, and testing the program. The person in charge of the keyboard and mouse is called the "driver." The other is called the "navigator" and is responsible for continuously reviewing what is typed. While the driver focuses on the tactical aspects of the current task, the navigator also considers the strategic direction of the work. The two programmers switch roles frequently.[2]

Some of the reported advantages of pair programming in organizations are that

- the design quality improves;
- the defect rate decreases;
- the team solves problems faster;
- the partners learn from each other;
- people learn to work together;
- people enjoy their work more.[3]

With these types of benefits it wasn't long before a few instructors of introductory programming classes started using the pair programming technique as a teaching method. On the surface this seems to make sense. The partner model has been used with great success in other academic disciplines. In the natural sciences, students almost always work with a partner in the laboratory.[4] In addition, this model fits in well with the millennial students desire to work and learn in teams.[5,6] However, most instructors still require students to complete programming assignments independently. The concern is that a least one of the partners will not learn as much as if he or she would have done the assignment alone. Yet a number of studies have shown that the use of pair programming in introductory programming classes improves course completion and retention in the computer-related majors.[7,8,9] Plus, these students were more likely to pass subsequent programming courses that required them to program solo.[7]

But what happens when students who learn to program using the solo approach are exposed to pair programming later? In this paper I discuss the results of using pair programming assignments in a web programming course. Most of the students in the course were at the junior or senior level and none of them had been formally introduced to pair programming prior to taking the course. I was especially curious to find out how the students would react to being forced to work with a partner when nearly all of the assignments in this class and previous classes have been coded individually.

Methodology

I have assigned pair programming assignments for several semesters of the web programming course. However, the fall of 2008 is the first time I started to formalize the process and analysis of the results. I introduced the students to pair programming by using part of a class period to describe the technique. We then discussed the benefits and drawbacks to organizations using this approach. I wanted students to understand that this is a legitimate software development method. The idea was that this would increase their desire to put effort into the pair programming assignments. I let students choose their own partner and made sure they understood that they were only to work on the project with the partner.

The project itself was challenging, but could be done with techniques that had already been applied in previous assignments. This allowed students to focus more on the logic of the program rather than on learning new language constructs. The students had one week to complete the project. The students were given time during class to work with their partner. However, students still had to find additional time outside of class to work on the assignment.

The fall of 2008 students were given a survey asking questions about their experience. The questions were in the areas of peer evaluation, comparison of pair programming to solo programming, and suggestions for future projects. Student feedback in earlier semesters was collected by less formal class discussions

In addition, the programs were compared to individually produced programs. I looked at design quality, number of defects, and code style consistency.

Students Perspective

The survey used a Likert scale where students rated their agreement with questions as 1. Strongly Disagree; 2. Disagree; 3. Not Sure, 4. Agree; or 5. Strongly Agree. The students also had the opportunity to answer some open-ended questions. Because there were only 10 students in the class in the semester that the survey was given, I did not do a statistical analysis. However, there were some definite trends and the survey seemed to match closely with student feedback from other semesters. The following are the results of the fall 2008 survey. Nine out of ten of the students completed the survey.

Were you and your partner able to reduce the program errors and bugs more quickly than working individually?

Most of the students agreed they were able to more debug the program quicker when working in partners (8 of 9). The one student that disagreed with this statement is an "A" student. However, he typically approaches and designs his programs in a different manner than most of the other students. There were several comments similar to, "two sets of eyes help cut down on simple mistakes."

Were you able to learn new techniques and approaches from your partner?

Seven agreed, one was neutral and one disagreed with this statement. These results were consistent with comments made during discussions and on the survey such as, "...being able to see problems from different perspectives." This also matched the results of the survey question, "Did you learn more working with a partner than you would have individually?"

Were you able to solve design problems more easily as partners?

The results were exactly split on whether or not it was easier to solve design problems as partners (4 agreed, 4 disagreed and 1 not sure). The results on question may be improved with more pair programming experience. Learning how to work together most likely created some overhead that made solving the problem with a partner more difficult in some cases.

Were you better able to stay on task while working with a partner rather than individually?

The seven students that agreed with this statement all strongly agreed. In fact, this may be one of the biggest benefits of pair programming in general, not just as a teaching tool. Following are some of the comments from discussions and the survey.

> *"It kept me on schedule. I also wouldn't walk away from the project when I became frustrated."*
>
> *"Helps with time management."*
>
> *"We were able to finish after working around 3 hours. This is much less than what it would have taken individually. My focus on the project shot up greatly when I knew someone else was counting on me. It also made programming fun."*
>
> *"Helps you stay on task and get it done. I think you work more intensely."*

Did you and your partner get along well while working on this project?

All the students got along well with their partners. This may be a result of letting students pick their own partners. In addition, this was a small class so by the time they did the pair programming project they already knew each other pretty well.

Of course there were drawbacks to working together. Most of the comments in this area were related to trying to schedule a time to work together. For example,

> *"We both work and have conflicting schedules."*

> *"I kept wanting to work on it but couldn't since I had to be with my partner."*

There were also comments related to what I would consider the process of learning to work with a partner.

> *"Sometimes one person has own agenda and take the lead more often than the other."*

> *"I felt my partner was mildly shy and wouldn't be forthcoming on his ideas."*

Teachers are always looking to find ways to get student engaged. From that standpoint it was nice to have several students say that the pair approach made programming fun.

Instructor Perspective

I analyzed 10 pair programming projects from fall 2006, 8 projects from fall 2007, 4 projects from fall 2008, and 5 projects from fall 2009. The programs were compared to individually produced programs in the areas of design quality, number of defects, and code style consistency.

In the area of design quality I looked for things like modularity and efficiency of the code. I found little difference between the individual and pair produced programs. This was especially true for the "A" and "B" students. For the poorer students, there was a definite improvement when paired with a better student. In the few cases where poorer students were paired together, what was most significant was that they were more likely to complete the whole project. Each semester I assigned relatively small projects using constructs they should have already been familiar with. More complicated projects may produce more significant differences in design quality.

Not surprisingly, the results for number of defects were very similar to those of design quality. The number of defects didn't vary greatly between individual and pair projects for the better students. The poorer students benefited the most from the pair experience. They were much more likely to attempt and complete all of the program specifications.

The area where I saw the most improvement was in code style consistency. I looked at things such as consistent and appropriate indentation of the code, use of meaningful variable names, and commenting of the code. This improvement may be due to the fact that the student in the driver role was conscious of the need to keep the code easy to read for the observing student.

Another factor was likely that the students spent more time designing a solution before jumping in and typing. It was easier to be consistent when the students knew where they were headed.

The most encouraging aspect of the pair programming projects from my perspective was the level of student engagement. It was obvious from observing students working in the computer lab that they were much more engaged in the project. Generally speaking, they were focused and actively discussed the project. Just from the looks on their faces I could see they were enjoying the work more than when working on projects individually. The student comments and survey results confirmed that observation. Students working in pairs did not have a chat or email window open like many of them do when working alone. In addition, no one had headphones on and was listening to music when working with a partner. Working with a partner seemed to improve focus.

Conclusions

Though this study was very limited in scope, it certainly seems to be consistent with the results of studies done at other universities. The benefits of pair programming seem to hold true for students in a web programming class, despite their lack of exposure to pair programming in introductory programming courses.

In my opinion, the most important finding was the impact that pair programming has on student engagement. This most likely would apply to any well constructed pair learning exercise. For one, this generation of students generally enjoys working with other students. Plus, working with a single partner forces the student to focus and be an active participant. In a world with so many distractions and students so used to multitasking, the benefits of having students direct their whole attention on a task opens doors to deeper levels of thinking. The challenge as a teacher is to figure out how to expand the use of pair programming projects and pair learning in general without creating a burden on students with busy schedules.

Bibliography

1. Beck, Kent, *Extreme Programming Explained*, Addison-Wesley, 1999.
2. Williams, Laurie, Kessler, Robert R., Cunningham, Ward, and Jeffries, Ron, Strengthening the Case for Pair-Programming, IEEE Software, Vol. 17, No. 4, July/August, 2000.
3. Cockburn, Alistair and Williams, Laurie, The Costs and Benefits of Pair Programming, *Proceedings of the First International Conference on Extreme Programming and Flexible Processes in Software Engineering*, 2000.
4. Williams, Laurie and Layman, Lucas, Lab Partners: If They're Good Enough for the Natural Sciences, Why Aren't They Good Enough for Us?, *Conference of Software Engineering Education and Training 2007*, Dublin, Ireland.
5. Raines, Claire, Managing Millennials, 2002, http://www.generationsatwork.com/articles/millenials.htm, accessed December 6, 2008.
6. Oblinger, Diana and Oblinger, James (Eds.), *Educating the Net Generation*, Educause, 2005.
7. McDowell, Charlie, Werner, Linda, Bullock, Heather E., and Fernald, Julian, Pair Programming improves student retention, confidence, and program quality, Communication of ACM, Vol. 49, No. 8, August 2006.
8. Cliburn, Daniel, Experiences with Pair Programming at a Small College, *Proceedings of the Midwestern Conference of the Consortium for Computing Sciences in Colleges*, 2003.

9. Williams, Laurie and Kessler, Robert R., Experimenting with Industry's "Pair-Programming" Model in the Computer Science Classroom, Computer Science Education, Vol. 11, No. 1, March, 2001.

Biographical Information

TROY HARDING
Associate Professor
Engineering Technology Department
Computer Systems Technology
Kansas State University at Salina