AC 2009-2253: PAIR PROGRAMMING IN A CAD-BASED ENGINEERING GRAPHICS COURSE

Robert Leland, Oral Roberts University

ROBERT LELAND received a S.B. in Computer Science from MIT in 1978, a M.S. in System Science from UCLA in 1982 and a Ph.D. in Electrical Engineering from UCLA in 1988. From 1989-1990 he was a visiting assistant professor at the University of Minnesota. From 1990-2005 he served on the faculty at the University of Alabama in Electrical and Computer Engineering. Since 2005 he has served on the faculty at Oral Roberts University in Engineering and Physics. His research interests include controls, MEMS, and engineering education.

Pair Programming in a CAD Based Engineering Graphics Course

Abstract

Pair programming was introduced into a course in engineering graphics that emphasizes solid modeling using SolidWorks. In pair programming, two students work at a single computer, and periodically trade off roles as driver (hands on the keyboard and mouse) and navigator (discuss strategy and design issues). Pair programming was used in a design project, and in a subsequent year in a design project and several smaller special projects. Student outcomes for two years were compared with a previous year in which pair programming was not used. Improvements were seen in design project scores, overall course scores, and project submission rates. The course is normally taken by first year students during the spring semester. Retention into the sophomore year was also higher for students participating in pair programming.

1. Introduction

Pair programming, an ingredient in extreme programming, has been used extensively in software development in industry, and has been used experimentally in computer programming based courses for engineering students. This paper describes the introduction of pair programming into the course EGR 140 Engineering Graphics at Oral Roberts University. The course uses the CAD software SolidWorks, and emphasizes solid modeling. Pair programming was introduced in a design project and several smaller special projects.

In pair programming, two students work on the same computer, and share one keyboard and one mouse. One student is the driver, and is operating the keyboard and mouse. The driver is actually creating the solid models. The other student is the navigator, who is checking to see that the specifications in the assignment are being met, thinking about the next step, and giving advice.

Pair programming is a part of a larger software development process known as Extreme Programming (XP), which has been reported to improve morale and customer satisfaction, and reduce project schedules[8]. The components of XP can be used to detail an educational process to develop expertise in software design[9]. A study involving 1200 students in introductory programming classes at two universities showed that students who engaged in pair programming performed as least as well as students working independently. A greater percentage of paired students passed the course with a grade of C or better. Also, a much larger percentage of the paired students declared a Computer Science major one year later[8]. In a study examining student behavior in computer labs, focus groups revealed that the paired students appreciated the ability to get quick answers to questions, without having to wait for an instructor. In addition, the lab instructors felt pair programming made their jobs easier as well[7]. In one study, the use of pair programming significantly increased the percentage of female students choosing to major in Computer Science[5]. Verbal responses from middle school girls involved in pair programming showed it was well received[5]. Students using pair programming were more likely to turn in working programs, were more likely to turn in their assignments to begin with, and reported being more confident and more satisfied with their experience[4]. In another study of pair

programming in an introductory C++ programming course, feedback from instructors indicated that students completed assignments in less time, and overcame roadblocks such as syntax errors more quickly. Student feedback also indicated that pair programming was an effective learning experience. Students also felt more confident, and that the quality of their work was better. Students felt the assignments were less stressful, and the instructors also observed a more positive and less stressful atmosphere in the class[2]. Suggested guidelines for pair programming classes include pairing students by skill level, making lab sessions that use pairing mandatory, scheduling so assignments can be mostly finished in session time, and creating a collaborative environment[1].

The use of pair programming in educational contexts has been reported primarily in introductory programming courses[1,2,4,7,8]. Pair programming was used in a Computer Architecture course. Student feedback indicated this was a positive experience, and student performance was in line with or better than that of students who worked independently[3]. This paper describes the author's experience in extending pair programming beyond the traditional computer programming context, and employing it in an Engineering Graphics class. Student performance and retention before and after the introduction of pair programming are compared.

2. Pair Programming in Engineering Graphics

This paper describes the introduction of pair programming into the course EGR 140 Engineering Graphics at Oral Roberts University. The course teaches the use of SolidWorks in creating solid models, assemblies, and drawings of those models. The approach is primarily learning by doing with small amounts of instruction, modeling and coaching. Pair programming was introduced through special design projects. The students worked individually on the majority of in-class work and homework assignments, as well as all tests. Thus students worked individually in acquiring basic skills, and worked in pairs when applying those skills to more challenging and open ended problems.

The solid modeling task is very different that writing a computer program, since a procedural object is not being produced and no new data structures must be designed. The solid modeling task shares aspects with programming, such as the need for conceptualization, identification of a process for creating a solid part, the limitations created by early design decisions, etc. Roadblocks in using the software due to student errors, similar to syntax errors, are also common and must be overcome. Significant differences in the tasks also exist. Rather than a sequence of instructions, a sequence of steps is identified to create the object. The creation of the objects and assemblies requires some common sense, planning and problem solving in selecting a process for creating the parts. In general, the product produced in solid modeling is less complex and more transparent than a computer program, so errors are easier to detect. Also, there is usually instant visual feedback telling the student if their steps to create an object are correct or not. However for more complex objects and assemblies, the constraints created by a design choice are not always immediately obvious. It is probably the novice status of the students that contributes the greatest challenges, so pair programming may be most useful to learning, but may not ultimately be part of their professional practice.

As in programming classes, a very wide range of expertise is represented. In this author's experience, some students can complete an exam in 10 minutes that some students will not manage to complete in 50 minutes. The idea of thinking ahead, planning, and making good initial design decisions is not innate to most students, and must be learned. Also, students working in pairs can be constrained to use a single computer, keyboard and mouse. In solid modeling, the mouse is used in a more 'analog' manner to create various shapes and dimensions. In general dimensions are entered by keyboard.

Although the students frequently do not interact in strict driver-navigator roles, this is the ideal presented. The students are to alternate roles. In class this occurs at fixed intervals of time. In industrial practice, this alternation frequently depends on which programmer is implementing their idea, and which is giving feedback.

Pair programming was introduced into an engineering graphics course normally taken by first year students in the spring semester. The course carries two credit hours, and meets for three hours per week. The students represent engineering majors, with concentrations in mechanical, electrical and computer engineering, biomedical engineering majors, and physics majors. A majority of the students are in the mechanical engineering concentration. A small number of students from computer science and other majors have also taken the course. The students typically have diverse backgrounds with respect to computer expertise, and intuition about solid objects, drawings and assemblies.

Pair programming was introduced in two consecutive years, 2007 and 2008. The first year, pair programming was limited to a single major project that was originally allotted four class periods. The second year, pair programming was used in the major project and several new smaller projects, which were allotted two class periods each. The remaining in class exercises and homework assignments, as well as all tests, were completed by the students working individually.

Pairs were selected by the instructor. Whenever possible, female students were paired together, and students were paired with other students of similar ability. The similar ability pairing was done in order to ensure participation by both students in the pair. While working in class, students were instructed to switch roles every 10 to 20 minutes. The times to switch were announced by the instructor. There were some students who did not switch roles at these times, and there was one pair where only one student attended the class. The instructor was not able to monitor how the students interacted outside of class. For most of the smaller projects, which required one to two class periods, the students were able to complete most or all of the project during the scheduled class periods. For the design project, which was more involved, a significant amount of the work was performed in four to five class periods, although more out of class work was involved.

In 2006 and 2007, the major design project consisted of a precisely specifed shaft, flywheel, mount, baseplate and bearing model that the students must create in SolidWorks. In 2008, the major design project was to create a model of a locomotive engine with working pistons that would drive the wheels based on photos and a diagram of the linkage between the wheels and pistons. In 2008, several smaller projects using pair programming were also assigned. In

general these also required the students to model a solid object or assembly from a photo. The assignments are described in the appendix.

3. Results

Several effects were noticed by the instructor when pair programming was introduced. First, this introduced teams into the course, which made it more 'relational', which in general created a positive environment for first year students that should support retention. Secondly, the percentage of projects that were turned in on time increased. Third, the percentage of students who seemed 'lost' was reduced. Fourth, students seemed to enjoy the class more and interact more like professionals.

Students were considered to have not significantly participated in the class if they did not attempt the final two exams and the design project. In general these students did not attempt other exams, turn in homework, or attend class. These students are excluded from the results for the design project and course scores and the retention study.

In Spring 2006, prior to introducing pair programming, the average score on the major design project was 71.26 out of 100. Three out of 26 students did not turn in the project. Their scores (0) are not included in the average. After introducing pair programming, the average score increased to 86.2 in 2007 and 86.6 in 2008. In 2007 and 2008, all students who significantly participated in the course turned in the design project. It should be noted that the design project was the same for 2006 and 2007, and a more advanced design project was assigned in 2008. This data is summarized in Table 1.

Overall course scores for the students for 2006 – 2008 were comparable. After excluding students who did not significantly participate (one student in 2006, one student in 2007, and one student in 2008), the average over all scores were: 2006: 79.9, 2007: 84.81, 2008: 84.9. This data is shown in Table 1. Slight increases are seen in the years using pair programming, but the number of students is too small for these differences to be statistically significant. The comparability of results does indicate that pair programming was not hurting the students. This is consistent with other results for pair programming reported in the literature.

Semester	Spring 2006	Spring 2007	Spring 2008
Total Students Considered	26	22	26
Average Design Project Score	71.3	86.2	86.6
Students Not Turning In Project	3	0	0
Overall Course Score	79.9	84.81	84.9

Table 1. Student Design Project Scores.

During the same period of time, retention improved dramatically. The list of students enrolled in EGR 140 in the Spring semester was compared to the class roster for a mandatory departmental seminar in the following Fall. Students who enrolled in the seminar and attended more than one seminar, or who otherwise were known to still be in the program, were considered to be retained. Students who were juniors and seniors in EGR 140, or who were retaking EGR 140, or who did

not significantly participate in EGR 140 were excluded. Two students in Spring 2006 who were Computer Science majors were excluded. Transfer students in their first year at ORU were included in the retention study. The retention rates are indicated in Table 2 below.

Semester:	Spring – Fall 2006	Spring-Fall 2007	Spring-Fall 2008
Total Students Considered	23	21	25
Students Retained	13	19	19
Percent Retained	57%	90%	76%

Table 2. First to Second Year Retention of Students taking EGR 140.

Although the sample size is fairly small, this is a large increase in retention of students into the sophomore year, which is a key retention barrier. Although there may be other factors involved in the increased retention, it appears that the use of pair programming certainly did not hurt retention. Note that the students considered here are first year students participating in EGR 140, which is taught in the Spring semester, rather than all students entering in the Fall semester. Also the department offers a program in Engineering Physics, and a small number of students in EGR 140 are Physics majors, and these students are also included in the above results.

Although two sections of the course are taught every spring, they do not provide a good basis for comparison and assessment, since one of the classes consists primarily of calculus ready first year students, and the other non-calculus ready students. Therefore the comparison is made between students taking the course before and after the introduction of pair programming.

4. Conclusion

Pair programming can be used in an Engineering Graphics course, and appears to positively influence student performance. In addition, higher levels of retention were seen after pair programming was introduced. The instructor intends to continue using pair programming in this course, and will attempt to improve student compliance in alternating roles.

Bibliography

1. J. Bevan, L. Werner, C. McDowell, 'Guidelines For the Use of Pair Programming In a Freshman Programming Class,' *Proceedings of IEEE-CS Conference on Software Engineering and Training*, 2002.

2. S. F. Freeman, B. K. Jaeger, J. C. Brougham, 'Pair Programming: More Learning and Less Anxiety in a First Programming Course,' *Proceedings of the ASEE Annual Conference and Exposition*, June 2003.

3. E. F. Gehringer, 'Is Pair Programming an Effective Way To Teach Computer Architecture?' *Proceedings of the ASEE Annual Conference and Exposition*, June 2003.

4. B. Hanks, C. McDowell, D. Draper, M. Krnjajic, 'Program Quality with Pair Programming in CS1,' *Proceedings of the 9th Annual Conference on Innovative Technology and Computer Science Education,' June 28–30, 2004, Leeds, United Kingdom*, 2004.

5. L. L. Werner, J. Denner, S. Bean, 'Pair Programming Strategies For Middle School Girls,' *Proceedings of the Seventh IASTED International Conference Computers and Advanced Technology in Education, August 16-18, 2004*, p. 161-166, 2004

6. E. N. Wiebe, L. Williams, J. Petlick, N. Nagappan, S. Balik, C. Miller, Miriam Firzli, 'Pair Programming In Introductory Programming Labs,' *Proceedings of the ASEE Annual Conference and Exposition*, June 2003.

7. L. Williams, C. McDowell, N. Nagappan, J. Fernald, L. Werner, 'Building Pair Programming Knowledge through a Family of Experiments,' *Proceedings 2003 International Symposium on Empirical Software Engineering, Sept. 30 – Oct. 1, 2003. Rome, Italy*, 2003.

8. L. Williams, R. Upchurch, 'Extreme Programming For Software Engineering Education,' *Proceedings of the 31st ASEE/IEEE Frontiers In Education Conference, Oct. 10-13, 2001, Reno, NV*, 2001.
9. C. McDowell, L. Werner, H. Bullock, & J. Fernald, The impact of pair programming on student performance, perception, and persistence, *Proceedings of the 25th International Conference on Software Engineering, Portland, OR, 2003*, 602-607.

10. Mid-continent Railway Museum, New Freedom, Wisconsin, www.midcontinent.org .

11. Steam Locomotive Walshaert Valve Gear Diagram, home.roadrunner.com/~trumpetb/loco/wdiagram.html .

Appendix: Design Projects Used.

Design Project for Spring 2006 and Spring 2007: Develop a SolidWorks assembly for the pulley, shaft and mounting below.

Dimensions are completely specified, and the wheel and shaft are free to rotate.

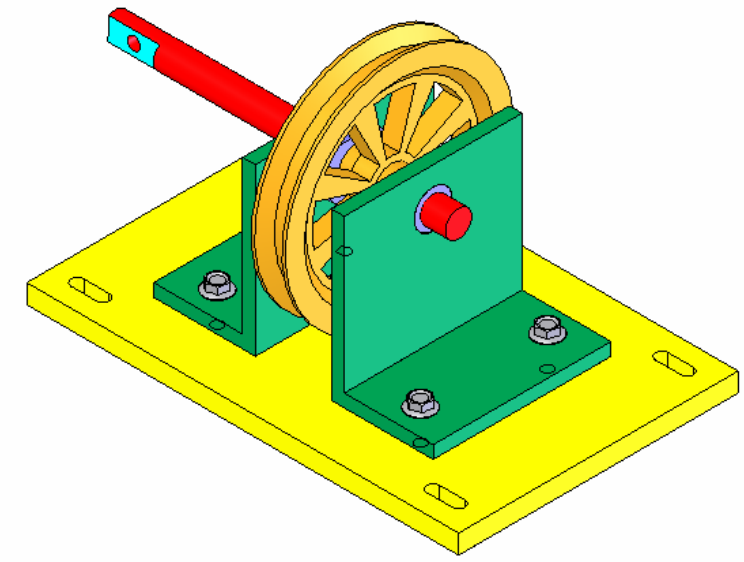

Figure 1. Assembly used for design project in Spring 2006 and Spring 2007. Courtesy of Dr. Richard Martin, PhD, PE, Aztec Engineering.

Design Project for Spring 2008: Develop a SolidWorks assembly for a steam locomotive engine.

Partial instructions:

- You should ignore all of the detailed piping on the boiler.
- All bolts may be omitted.
- Clear glass (set material) window panes must be used on the cab. They do not have to slide or open.
- The drive mechanism should be included, except the numbered items in the diagram on page 2 (Figure 3.) may be omitted. Note that part of the drive mechanism is missing from the train in the picture.
- The headlight may be a simple circular shape on the front of the tank, and need not be clear. Extra credit will be given for more realistic headlights.
- Both pistons should be included, and the lower one should work when the wheels are turned.
- The large wheels may be identical, and should all have spokes, like the first and third.
- The offset weights (crescents) on the large wheels may be identical for all three wheels, and similar to the rear wheel (flat on one side). They should be located opposite the connection of the wheel to the drive mechanism.
- The smokestack should have an opening at the top.

.

Figure 2. Locomotive photographs used in the Spring 2008 design project. Photographs by Paul Swanson, courtesy of Mid-Continent Railway Museum, North Freedom, Wisconsin.

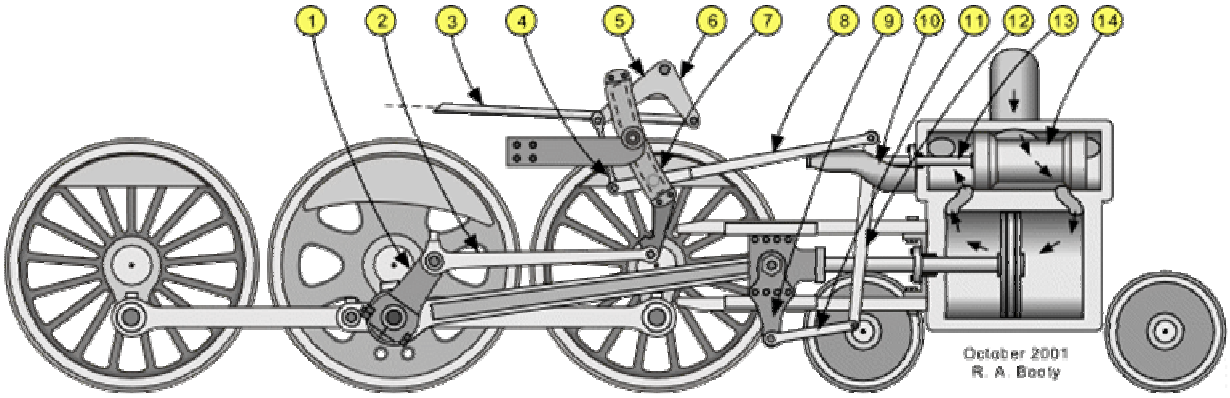

Figure 3. Locomotive drive mechanism used in the Spring 2008 design project. Diagram is from Robert Booty's Steam Locomotive Valve Gear website: http://home.roadrunner.com/~trumpetb/loco/ , used with permission.

Other Projects in Spring 2008:

- Lid from a CD case. Students examine CD case lids and create 3D models.
- Mount for a dish antenna. Students create a 3D model of an antenna mount from a photograph.
- Can opener assembly. Students examine a hand operated can opener and create an assembly model.
- Electrical conduit box fabricated from sheet metal. Students examine conduit boxes and create 3D models.

More detailed information on the projects is available on request.