Pair Programming in an Introductory Computer Science Course: Initial Results and Recommendations

Laurie Williams
Department of Computer Science
North Carolina State University
Raleigh, NC 27695
+1 919-515-7925
williams@csc.ncsu.edu

Kai Yang
Department of Computer Science
North Carolina State University
Raleigh, NC 27695
kyang@unity.ncsu.edu

Eric Wiebe
Department of Math, Science and Technology Educ.
North Carolina State University
+1 919-515-1753
wiebe@unity.ncsu.edu

Miriam Ferzli
Department of Math, Science and Technology Educ.
North Carolina State University
+1 919-828-5599
@unity.ncsu.edu

Carol Miller
Department of Computer Science
North Carolina State University
+1 919-515-2042
miller@unity.ncsu.edu

ABSTRACT

Prior research indicates that pair programming, whereby two programmers work collaboratively on the same design, algorithm, code, or test, produces higher quality code in essentially half the time taken by solo programmers. An experiment was run at North Carolina to assess the efficacy of pair programming in the introductory CS1 course. Results indicate that relative to students who program individually, pair programmers are more self-sufficient, perform better on projects, and are more likely to complete the class with a C or better

Keywords

pair programming, collaborative learning, Computer Science education, Extreme Programming, XP

1. INTRODUCTION

In industry, programmers collaborate for the majority of their day. In *Peopleware* [5], it was reported that software developers generally spend 30% of their time working alone, 50% of their time working with one other person, and 20% of their time working with two or more people. Yet, most often when completing their degree, programmers must learn to program alone; collaboration is considered cheating. This is unfortunate not only because collaboration is encouraged and required in a student's future professional life, but there are also findings that cooperative and collaborative pedagogies are beneficial to students [8, 9].

An emerging software development methodology, Extreme Programming (XP) [1], has recently popularized a structured form of programmer collaboration called pair programming. 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*, is typing at the computer or writing down a design. The other partner, called the *navigator*, has many jobs. One is to observe the work of the driver – looking for tactical and strategic defects in the work of the driver. Tactical defects are syntax errors, typos, calling the wrong method, etc. Strategic defects are when the driver is headed down the wrong path – what they are implementing just won't accomplish what it needs to accomplish. The navigator is the strategic, longer-range thinker. Any of us can be guilty of straying off the path. A simple, "Can you explain what you're doing?" from the navigator can serve to bring us back to earth. The navigator has a much more objective point of view and can better think strategically about the direction of the work. Additionally, the driver and the navigator can brainstorm on-demand at any time. An effective pair programming relationship is very active. The driver and the navigator communicate, if only through utterances, at least every 45 to 60 seconds. Periodically, it's also very important to switch roles between the driver and the navigator. Research results [3, 12, 13] indicate that pair programmers produce higher quality code in about half the time when compared with solo programmers. Those who follow the XP methodology feel so strongly about the benefits of pair programming that all production code must be written with a partner [11]. Even prototyping done solo is scrapped and re-written with a partner.

The research results referenced above were based on experiments held at the University of Utah in the senior-level Software Engineering course [12-15]. The focus of this research was on the affordability of the practice of pair programming, the ability of the practice to yield higher quality code without significant increases in time/cost. However, the researchers observed educational benefits for the student pair programmers. These benefits included superior results on graded assignments, increased satisfaction/reduced frustration from the students, increased confidence from the students on their project results, and reduced workload of the teaching staff.

These observations inspired further research directed at the use of pair programming in educating Computer Science students. Educators at the University of California-Santa Cruz have reported on the use collaborative laboratory activities in an introductory undergraduate programming course, specifically in the form of pair programming [2, 7]. They have found that pair programming improved retention rates and performance on programming assignments. This paper details the results of an additional experiment that was held at North

Carolina State University (NCSU) in 2001. The experiment was specifically designed to assess the efficacy of pair programming in an introductory Computer Science classroom.

2. EXPERIMENT

An experiment was conducted in the CS1 course, Introduction to Computing – Java. The course is taught with two 50-minutes lectures and one three-hour lab each week. Students attend labs in groups of 24 with others in their own lecture section. The lab period is run as a closed lab, whereby students are given a weekly assignment to complete during the allotted time. The lab assignments are "completion" assignments whereby students fill in the body of methods in a skeleton of the program prepared by the instructor. Student grades are based on two midterm exams, one final exam, lab assignments, and three programming projects that are completed outside of the closed lab. The projects are generative, in that students start from scratch without any structure imposed by the instructor. The course is a service course, and is therefore taken by many students throughout the university. Most students are from the College of Engineering. Additionally, most students are freshman; however students of all undergraduate and graduate levels also take the course to learn competitive programming skills.

As educators, we were concerned that the academically weaker or less motivated students would allow their partner to do all the work. To alleviate this concern, each time the students were assigned a new partner, they were required to complete a peer evaluation on their prior partner. The students rated their partner on a scale from 0 (poor) to 20 (superior) for each of these five questions:

1. Did your partner read the lab assignment and preparatory materials before coming to the scheduled lab?
2. Did your partner do their fair share of the work?
3. Did your partner cooperatively follow the pair programming model (rotating roles of driver and navigator)?
4. Did your partner make contributions to the completion of the lab assignment?
5. Did your partner cooperate?

The sum of the ratings on each of these questions yielded a grade from 0-100%. Each student's lab grade was multiplied by this peer evaluation factor. For example, if a student had a 90% lab average but a peer evaluation score of 50%, they received a final lab grade of 45%. We had successfully used this form of peer evaluation in past classes. We have found that it does motivate students to do their share of the work. In general, 95% of the class will report that their partner did their share of the work, and would assign him or her 100%. In a minority of cases, the peer evaluation score is a strong signal of a student who is truly not putting forth the necessary effort.

Closed labs are excellent for controlled use of pair programming [2]. The instructor or teaching assistant can ensure that people are, indeed, working in pairs at one computer. He or she can also monitor that the roles of driver and navigator are rotated periodically. Many classes have programs that require work outside of close lab. We have found that you simply cannot enforce pair programming outside of the classroom. Some students will come to fully appreciate the benefits of pair programming and will seek to work with a partner outside of class. Others will choose to work alone, whether they prefer pair programming or not, often because they would prefer to work on homework in the comfort of their dorm room at any hour of the day or night. Pairing outside of lab requires time planning and coordination; some students view the added coordination as a hassle. In the CS1 experimental course, the students completed three programming projects outside of the closed lab environment. We gave the students in both sections the option of working alone or pair programming for these projects. Many chose to pair program. However, we found instances of students who were doing very poorly in the class pairing with students who were doing well in the class. Often, these students did well on the projects, causing suspicion that the stronger student did most or all of the work. As a result, starting in the Spring 2002 class, we instituted a policy that students must earn the right to pair program on the projects by attaining a score of 70% or better on the exams.

The Fall 2001 experiment was run in two sections of the course; the same instructor taught both sections. Additionally, the midterm exams and the final exam were identical in both sections. (The exams were given to the second section immediately after the first, leaving little time for students to tell the second section about the exam content.) One section had traditional, solo programming labs. In the other section, students were required to complete their lab assignments utilizing the pair programming practice. When students enrolled for the class, they had no knowledge of the experiment or of that one section would have paired and other would have solo labs. In the pair programming labs, students were randomly assigned partners based on a web-based computer program and not student preferences. They worked with the same partner for two to three weeks. If a student's partner did not show up after 10 minutes, the student was assigned to another partner. If there were an odd number of students, three students worked together; no one worked alone.

In the Fall, 112 students were in the solo section and 87 were in the paired section. Our study was specifically aimed at the effects of pair programming on beginning students. Therefore, we chose to analyze the results of the freshman and sophomores only. We also only chose students who took the course for a grade, concluding that students who audited the class or took it for credit only were not as motivated to excel as other students. This reduced our sample size to N=69 in the solo section and N=44 in the paired section. In our experiment, we examined the following hypotheses:

- A higher percentage of students that have participated in pair programming in CS1 will succeed in the class by completing the class with a grade of C or better.
- Students' participation in pair-programming in CS1 will lead to better performance on examinations (exams are completed solo by all students) in that class

- Students' participation in pair-programming in CS1 will lead to better performance on course projects in that class
- Students' participation in pair-programming in CS1 will lead to a more positive attitude toward the course and toward Computer Science in general
- Students' participation in pair-programming lead to a lower workload for course staff

3. QUANTITATIVE FINDINGS

3.1 Success Rate/Retention

We examined the percentage of students who succeeded in the class by completing the course with a grade of C or better. Historically, beginning Computer Science classes have poor success rates. For all the good intentions and diligent work of computer science educators, students find introductory computer science courses very daunting—so daunting that typically one-quarter of the students drop out of the classes and many others perform poorly.

In the solo section, only 45% of the students we studied successfully completed the course with a grade of C or better. Comparatively, 68% of the students in the paired section met these criteria. A Chi-Square test reveled that this difference in success rates is statistically significant ($?^2(1)=7.056$, $p < 0.008$). These results are consistent with a similar study at the University of California UC-Santa Cruz that reported 92% of their paired class and 76% of their solo class took the final exam [7].

3.2 Performance on Examinations

On average, students in the paired section performed better on the two midterm examinations and the final examination, as shown in Table 1. We removed any scores of 0 from our analysis; these results are based on scores of students who attempted to take the exam.

Table 1: Exam Scores

Exam	Paired Mean	Paired Standard Deviation	Solo Mean	Solo Standard Deviation
Midterm 1	78.7	11.8	73.4	13.8
Midterm 2	65.8	24.2	49.5	27.2
Final	74.1	16.5	67.2	18.4

As stated earlier, students chose their class section without knowledge of the experiment or pair programming. We had hoped that their random enrollment in the class would yield equivalent sample groups based on their SAT-M scores. Unfortunately, this was not the case. The students in the paired group had a mean SAT-M score of 662.10 while the solo group had a mean score of 625.43. The One-Way ANOVA revealed that this difference was statistically significant ($F(1.101)=5.19$, $p<0.018$). An ANCOVA further revealed a correlation between SAT-M scores and exam scores, when considered for each exam individually ($F(1,98)=36.32$, $p<0.0001$; $F(1,98)=41.41$, $p<0.0001$). When using SAT-M as a covariate, an ANCOVA does not show any significant difference between sections with regards to midterm or final exam scores. Based on these results, we cannot conclude that pair programming in the laboratory helped students perform better on exams.

We wish to discuss two factors that may influence these results. First, changes in the lab portion of the course may have enough of an influence on student work and attitudes to keep them from dropping out of the course or boosting their grades enough to pass the course. This may have dampened the overall distribution of grades in the paired section since it may have kept poorer performing students in the calculation pool whereby these poorer performing students dropped the class or did not take exams, so they are not in the calculation pool. Researchers at UC-Santa Cruz have made this same speculation [7] because their paired section also did not have statistically significant higher test scores. Additionally, only approximately 40% of the exam content required program code to be written in the answers. The rest of the exams were short answer and multiple choice. Quite feasibly, pair programming might not help students answer short answer and multiple choice questions on the material better. The classes' scores on only the portion of the exam that did require programming is not available.

3.3 Performance of Programming Projects

On average, students in the paired section performed better on the two of three programming projects, as shown in Table 2.

Table 2: Programming Project Scores

Exam	Paired Mean	Paired Standard Deviation	Solo Mean	Solo Standard Deviation
Project 1	94.6	5.3	78.2	26.5
Project 2	86.3	19.7	68.7	33.7
Project 3	73.7	27.1	74.4	29.0

The ANCOVA demonstrated a statistically significant improvement in performance of the pairs on Project 1 ($F(1,94)=8.12$, $p<0.0054$) and Project 2 ($F(1,78)=4.52$, $p<0.0367$). However this demonstrated improved performance does not occur in Project 3. Perhaps, this is because by Project 3 the lower performing students had dropped in the solo section but were still working in the paired section.

3.4 Attitude

We hypothesize that students in paired labs will have a more positive attitude toward the course and about Computer

Science in general. We based this hypothesis on prior observations that beginning classes can be very frustrating. Students might debug a program for several hours because of a very simple syntactical error. These kinds of errors would likely be caught by the navigator, precluding the need for extensive, frustrating debugging sessions.

We looked at specific questions on the student course evaluation, because both sections were taught by the same instructor. The only data available on course evaluation is a mean score, so no statistical evaluation could be performed. On the course evaluation, a 1 is an unfavorable score and a 5 is a very favorable score. As shown in Table 3, students did feel more favorable toward the course and the instructor in the paired section:

Table 3: Course Evaluations

Exam	Paired Mean	Solo Mean
Course Effectiveness	3.97	3.58
Instructor Effectiveness	4.20	3.69
Classroom is Conducive to Learning	4.26	4.26

4. QUALTITATIVE RESULTS

In order to gain insights about the student-student, instructor-student dynamics of the pair-programming protocol, we collected observations during paired and unpaired computer programming laboratory sessions. Observational data was collected in the Spring semester 2002 on a weekly basis, during a continuation of the controlled study conducted in the Fall semester 2001. The setting for the observations was the actual laboratory CS1, which students attend for three hours every week. Analysis of the observational data allowed us to document major issues related to the pair-programming protocol that would not surface during the other components of the study.

4.1 Paired Labs

4.1.1. Learning

Without exception, students in the pair-programming lab sessions showed a high level of interaction with each other. Students were discussing issues related to the programming assignment on a consistent basis. Students questioned, directed, and guided each other throughout the lab session. When student pairs could not seem to answer questions on their own, they would ask the instructor; but the interaction with the instructor was usually very brief (less than five minutes). A lot of students' questions to instructors were of a logistical rather than conceptual nature. On a very frequent basis, pairs resolved their own problems without the instructor's help. Overall, instructors spent very little time answering questions. Most instructor-student interactions seemed to take the form of extended discussions. Students would want to know how to apply what they were doing to another scenario. Hypothetical discussions of applications showed evidence of higher-level thinking processes that went beyond the scope of the programming assignment.

4.1.2 Driver and Navigator Roles

Although the pair-programming protocol is set up to give students an opportunity to experience two different roles while programming, some students remain marginal players in this setting. Several students were observed to give little or no input during the entire lab session. According to cooperative/collaborative learning research, there are several reasons why students remain disengaged. Students may be mismatched based on their achievement level, gender, or cultural roles [10]. In such cases, it is common for one student in the group to take over.

With the exception of a few pairs, most student pairs seemed reluctant to reverse roles immediately after being told to. Some students remained in the same role—either navigator or driver—during the entire lab session. Other students reversed roles at times other than those indicated by the lab instructor. In either case, it seemed that students either found it inconvenient or unnecessary to reverse roles. Perhaps reversing roles at prescribed time interrupts the flow of work, so students opted not to reverse roles; or students need to find a pausing point before they can reverse roles.

Whether or not students take on their appropriate roles during the specified times, they do show increasing willingness to take on the driver/navigator roles with each passing week. Easing into the pair-programming protocol over time may suggest that students need time to become familiar with the paired protocol before they can feel comfortable. This makes sense since undergraduate students may not be accustomed to a collaborative learning approach in a computer lab session. Most students in this type of setting are used to individual work, even though they will encounter a collaborative approach to programming and project building in the workplace [4].

4.1.3 Instructors' Roles

The role of the laboratory instructor seems crucial to the success of the pair-programming protocol. When instructors explained and reinforced the pair programming protocol on a regular basis, students were more apt to assume appropriate roles as well as reverse roles when necessary. In labs where instructors forgot to ask students to reverse roles, no role reversal occurred. In labs where the instructor failed to enforce the pair-programming protocol, students opted for individual work. Students who chose to pair on their own did not follow the correct protocol. These students worked at their own computers while they engaged in some level of collaboration with no evidence of driver/navigator roles. Instructors that did enforce the pair-programming protocol, were more likely get students involved in team learning. Without instructor reinforcement, students very easily reverted to the individual work with which they are so accustomed.

4.2 Solo Labs

Overall, the solo labs, or control lab sessions, were very quiet. There was little or no discussion between students, and students had questions on a frequent basis. When the instructors answered students' questions, they spent a minimum of five minutes and a maximum of twenty minutes with each student. Instructors remained busy answering questions for the duration of the lab sessions. Often, students with questions sat and waited for long periods of time (maximum of thirty minutes) before they could get help. During this time, students seemed "stuck" and could not go any further. Some students in these situations opted to help each other, but their interactions remained brief and sporadic. On some occasions when they needed help but their neighbor seemed too busy to help them, students leaned over to look at their neighbor's computer screen.

4.3 Summary of Findings:

Students in paired labs engage in extensive discussion throughout the entire lab session, and students seem to help each other resolve questions. Instructors spend more time discussing advanced issues with students, rather than answering basic questions. Students seem to show evidence of higher order thinking—synthesizing and applying lab material to other scenarios. Students seem to take on navigator/driver roles with more ease when these roles are when reinforced by the instructor. Students switch roles at their own pace, perhaps because they need to reach an agreed "cut-off" point.

Students in solo labs seem to fall into student helper roles naturally. Students, who run into problems, spend a lot of time waiting for the instructor to help them. Although students seem to interact with each other, interactions are brief. Discussion between students is challenging, because students are at different points in their work and discussion would disturb their progress. Instructors seem to take on a very active role when helping students, rather than letting students figure things out on their own.

5. RECOMMENDATIONS

These benefits are not realized effortlessly. The teaching staff must be prepared to provide structure and guidance related to the practice. We now share our recommendation based on our experiences with pair programming [16]:

- Students cannot be expected to intuitively understand the pair programming practice. They need be educated in the subtleties of pair programming of the practices, such as the role of driver and navigator, the need to rotate roles, etc. Williams and Kessler [17] provide a useful reference for students; this should be augmented with an in-class discussion of the material.
- A structured hands-on pair programming tutorial, such as outlined in [16] and summarized as an Appendix at the end of this paper, will aid in the students realizing the value of pair programming.
- As stated earlier, pair programming works best in a closed lab setting. If this is not an option due to class structure, it is recommended that some class time is dedicated to allowing pairs to discuss their project, plans and meeting schedule. The more time the students have to bond and jell in class, the more likely their experiences outside of class are likely to be.
- In a closed lab, enforce the rotation of roles between driver and navigator by equipping the lab with a simple kitchen timer. The teaching staff should regularly remind the students to switch roles, but should allow for a reasonable delay for the students to finish their immediate task.
- In a paired lab, students ask far fewer questions. The teaching staff should resist the temptation to leave the students be. The teaching staff should follow the industrial practice of "Management by Walking Around." He or she should periodically circulate among the students, checking to see that neither student is dominating and both are contributing as equally as possible.
- Incorporate a peer evaluation system into your course practices and ensure the students' grades are impacted based on their peer evaluation.
- Several times throughout the semester, switch the makeup of the pairings. This serves several purposes. First, by the end of the semester, each student will have several sets of peer evaluation feedback, which is likely to be more representative of their overall contribution based on the opinion of several peers. Second, pairs are less likely to become irreconcilably dysfunctional because students know it is not a "permanent" arrangement. Lastly, students have the opportunity to learn from and get to know more people in their class.
- When utilizing pair programming in a class without a laboratory, consider making pair programming optional. Further, predetermine a level of performance required to "earn the right" to pair, preventing poor students from having a way to avoid learning necessary skills by pairing with a good student who might likely carry the workload of both.
- As much as possible, have different students start the lab as the driver each week. This can be done in creative ways, such as stating that the person with the shortest hair of the pair start as the driver one week, followed by the person with the longest hair of the pair in the following week.
- If there are an odd number of students in the class, have one three-person group before requiring any student to work alone. When everyone else in the room is collaborating, working alone feels exceedingly lonely.

6. SUMMARY

Previous anecdotal evidence supported educational benefits for the student pair programmers. These benefits included superior results on graded assignments, increased satisfaction/reduced frustration from the students, increased confidence from the students on their project results, and reduced workload of the teaching staff. A structured, empirical study began at NCSU in the Fall 2001 semester to qualitatively and quantitatively examine these anecdotal claims in a CS1 course; this paper outlines the results of one semester of this study. Our empirical results indicate that students who practice a pair programming technique in closed labs are more likely to persevere through CS1 and receive a grade of C or better. Student pairs also produced better grades on programming assignments. The students who are in the paired labs received higher grades, on average, on examinations, though this difference was not statistically significant. Qualitative observations supported that paired closed labs were a superior learning environment for the students. Paired labs are also less stressful for the teaching assistants because students are not as reliant on them as the sole provider of technical information and help.

What are the implications of these research results to Computer Science departments? Our findings support that CS1 closed labs would benefit from transitioning to paired labs. However, our results also caution that with the benefits realized with student pair programming come some costs. Teaching assistants must enforce the pair programming model by reminding students to periodically rotate the driver and navigator roles and to notice when one student might be dominating. In order to create an environment where all students have the opportunity and the incentive to participate in assignments, we recommend that partners are assigned and are not static for the entire semester. This provides a means for students to report via a peer evaluation on the contributions of their partners; these peer evaluations should have significant weight in a student's grade. Lastly, we advise that lower performing students are not given the opportunity to pair on assignments done outside of a closed lab. At NCSU, we have enforced a cut off that anyone who receives a grade of below 70 on the examinations must complete their outside projects solo.

7. ACKNOWLEGEMENTS

Funding for the research was provided by the National Science Foundation Grant 29728000.

8. REFERENCES

[1] Beck, K. (2000). Extreme Programming Explained: Embrace Change. Reading, MA: Addison-Wesley.

[2] Bevan, J., Werner, L., McDowell, C. (2002). Guidelines for the Use of Pair Programming in a Freshman Programming Class, *Fifteenth Conference on Software Engineering Education and Training (CSEE&T 2002).*

[3] Cockburn, A. & Williams, L. (2002). "The Costs and Benefits of Pair Programming" in *Extreme Programming Examined*, Boston: Addison-Wesley.

[4] Collings, P. & Walker, D. (1995). Equity issues in computer-based collaboration: Looking beyond surface indicators. In Proceedings of CSCL '95: The First International Conference on Computer Collaborative Learning, Schnase, J.L. , Cunnius, E.L., eds. Mahwah: Lawrence Erlbaum Associates, Inc., pp. 75-83.

[5] DeMarco, T., and Lister, T. (1987). *Peopleware*, New York: Dorset House Publishers.

[6] Fennema, E., & Sherman, J.A. (1976). Fennema-Sherman mathematics attitudes scales. Instruments designed to measure attitudes toward the learning of mathematics by females and males. JSAS: Catalog of Selected Documents in Psychology, 6(31), (Ms. No. 1225).

[7] McDowell, C., Werner, L., Bullock, H., Fernald, J. (2002). The Effects of Pair Programming on Performance in an Introductory Programming Course, Proceedings of the Conference of the Special Interest Group of Computer Science Educators (SIGCSE 2002).

[8] Slavin, R. (1980). Using Student Team Learning. Baltimore: The Center for Social Organization of Schools, The Johns Hopkins University.

[9] Slavin, R. (1990). Cooperative Learning: Theory, Research, and Practice. New Jersey: Prentice Hall.

[10] Webb, N.M., Nemer, K.M., Chizhik, A.W. (1998). Equity issues in collaborative group assessment: Group composition and performance. American Educational Research Journal, 35(4), 607-651.

[11] Wiki. (1999, June 29). Pair Programming. *Portland Pattern Repository*, Available at: http://c2.com/cgi/wiki?PairProgramming.

[12] Williams, L., Kessler, R., Cunningham, W., & Jeffries, R. (2000, July/August 2000). Strengthening the Case for Pair-Programming. *IEEE Software*, vol. 17, no. 3.

[13] Williams, L. A. (2000). *The Collaborative Software Process PhD Dissertation.* University of Utah, Salt Lake City, UT.

[14] Williams, L. A., & Kessler, R. R. (2000). The Effects of "Pair-Pressure" and "Pair-Learning" on Software Engineering Education. *Thirteenth Conference on Software Engineering Education and Training (CSEE&T 2000), pp. 59-65.*

[15] Williams, L. A., & Kessler, R. R. (2001, March 2001). Experimenting with Industry's "Pair Programming" Model in the Computer Science Classroom. *Journal of Computer Science Education, pp. 1-20.*

[16] Williams, L. A., & Kessler, R. R (2002), *Pair Programming Illuminated*, Addison Wesley.

[17] Williams, L. A. & Kessler, R. R. (2000), *All I Ever Needed to Know about Pair Programming I Learned in Kindergarten*, Communications of the ACM, May 2000.

APPENDIX: A PAIR PROGRAMMING TUTORIAL

The following three-step tutorial can be run in approximately one hour. There are three 15-minute activities with group discussion after each activity. In order to make the experience as enjoyable as possible for the students, you should have blank transparencies and markers available. Randomly, student groups should be chosen to share their drawings with the class. Students find the sharing enjoyable and entertaining. The sharing also aids in class discussion of the relevant points. Through this exercise, pairs learn that they can work together as a team by pairing because they work together, communicate, and have a superior knowledge of the overall team project.

Start the tutorial by having students form groups of 4. Determine a creative way for students to establish who is Student A, B, C, and D. We often do this by saying the person who woke up first is Person A, etc.

Activity 1: Individuals Working on a Team (15 minutes)

For the first activity, the group is shown a problem statement for a transportation device. The device needs to be able to:

- Transport people faster than they can move by walking, but must go less than 10 mph.
- Stop on demand.
- Carry at least one person.
- Restrain passengers, so they don't fall out.
- Look nice.

Each participant is given the assignment of completing one aspect of the transportation device design. The four roles are: Student A: Appearance, Student B: Propelling System, Student C: Braking System, Student D: Restraint System. Each participant must complete their assignment without collaborating with the team. Approximately two minutes are given for this activity. At this time, each participant is asked to draw the entire transportation device as they envision it, again without collaborating with team members (approximately 2 minutes). Lastly, the team members must integrate their design into one transportation device. They integrated device must take the appearance from the one participant in charge of appearance, the braking system from the participant that designed that, etc. This should take approximately 5 minutes. The remaining 6 minutes are spent on a discussion of how far people's individual view of the system was from the integrated device. Additionally, it is likely that the integrated device will not have components that do not logically fit together. Have selected student groups share their individual and integrated drawings.

The facilitator should point out the lessons of the exercise: When engineers work individually on a design, the components may not fit together when integrated. Additionally, individuals do not have a good feel of the overall design of the project.

Activity 2: Pair Programming (15 minutes)

For the second activity, the group is shown another, similar problem statement for a transportation device. The device needs to be able to:

- Transport faster than 10 mph, but slower than 100 mph.
- Stop on demand.
- Carry at least one person.
- Restrain passengers, so they don't fall out.
- Look nice.

Participants work in pairs to of complete two aspects of the transportation device design. The four roles are: Students A and B: Appearance and Propelling System, Student C and D: Braking System and Restraint System. Each pair must complete their assignment without collaborating with the other pair. Approximately four minutes are given for this activity. At this time, each participant is asked to draw the entire transportation device as they envision it, again without collaborating with any other team members – even their partner (approximately 2 minutes). Lastly, the team members must integrate their design into one transportation device. They integrated device must take the appearance from the one participant in charge of appearance, the braking system from the participant that designed that, etc. This should take approximately 3 minutes. The remaining 6 minutes are spent on a discussion. Again, have students share their individual and integrated drawings.

The facilitator should point out the lessons of the exercise: When engineers work pair on a design, the components are more likely to fit together better because they have indepth knowledge of two rather than only one aspect of the design. Pairs may be more creative in their designs as their synergized design is likely superior to that done alone.   Additionally, individuals have a better feel of the overall design of the project.

Activity 3: Pair Rotation (15 minutes)

Participants stay with the group of four they worked with in Activity III. For the next activity, the group is shown another, slighly different, problem statement for a transportation device. The device needs to be able to:

- Transport people faster than 100 mph.
- Stop on demand.
- Carry at least one person.
- Restrain passengers, so they don't fall out.
- Look nice.

Each participant is assigned a specific role in the design of the device. The four roles are: Student A: Appearance, Student B: Propelling System, Student C: Braking System, Student D: Restraint System. That particular participant is given <u>ultimate authority</u> of that aspect of the device in the upcoming collaborative effort. A rotation of pairs then takes place:

- Each participant is assigned a partner to work with. Together the pair work on the <u>two</u> things they were assigned for two minutes. (e.g. Students A and B; Students C and D)
- Partners rotate so that each person is paired with a team member they did not work with yet for the following two minutes. Together the pair work on the two things they were assigned for two minutes. (e.g. Students A and C; Students B and D)
- Partners rotate again so that each person is paired with the last team member they did not work with yet for the next two minutes. Together the pair work on the two things they were assigned for two minutes. (e.g. Students A and D; Students B and C)

At this time, each participant is asked to individually draw the entire transportation device, again without collaborating with team members (approximately 2 minutes). Lastly, the team members must integrate their design into one transportation device. They integrated device must take the appearance from the one participant in charge of appearance, the braking system from the participant that designed that, etc. This should take approximately 3 minutes. The remaining 4 minutes are spent on a discussion of how far people's individual view of the system was from the integrated device. Again, it is fun to make this point by having selected participant teams draw their own view and their integrated view on transparency slides.

The facilitator should point out the lessons of the exercise: Pairs that rotate around a group have a better understanding of the entire project. Additionally, the components that are designed by pairs that rotate around the group are more likely to fit together into a cohesive system.

Summary and Conclusion (5 minutes)

At the end of the session, the facilitator should re-point out the lessons learned from the exercise. A learning objective of the session was for people to experience pair programming first-hand. This was done through the activities. Added objectives of the activities were to show that pair programming can be used to better spread system knowledge around a group and to aid in individual components formulating one cohesive system when integrated. Additionally, participants learned about research

After reviewing the objectives of the exercise, the facilitator should ask for participant feedback and should allow an open discussion on implementing pair programming in the class. This kind of discussion could certainly take more than 5 minutes, but should be very valuable.

This tutorial is explained in further detail in [16].