Journal of Information Technology Education: Innovations in Practice Volume 15, 2016

Cite as: Dongo, T., Reed, A. H., & O'Hara, M. (2016). Exploring pair programming Benefits for MIS majors. *Journal of Information Technology Education: Innovations in Practice, 15,* 223-239. Retrieved from http://www.informingscience.org/Publications/3625

Exploring Pair Programming Benefits for MIS Majors

Tendai Dongo, April H. Reed, and Margaret O'Hara
East Carolina University, College of Business, Greenville, NC 27858

TendaiDongo@alumni.ecu.edu; reeda@ecu.edu; oharam@ecu.edu

Abstract

Pair programming is a collaborative programming practice that places participants in dyads, working in tandem at one computer to complete programming assignments. Pair programming studies with Computer Science (CS) and Software Engineering (SE) majors have identified benefits such as technical productivity, program/design quality, academic performance, and increased satisfaction for their participants. In this paper, pair programming is studied with Management Information Systems (MIS) majors, who (unlike CS and SE majors taking several programming courses) typically take only one programming course and often struggle to develop advanced programming skills within that single course. The researchers conducted two pair programming experiments in an introductory software development course for MIS majors over three semesters to determine if pair programming could enhance learning for MIS students. The program results, researchers' direct observations, and participants' responses to a survey questionnaire were analyzed after each experiment. The results indicate that pair programming appears to be beneficial to MIS students' technical productivity and program design quality, specifically the ability to create programs using high-level concepts. Additionally, results confirmed increased student satisfaction and reduced frustration, as the pairs worked collaboratively to produce a program while actively communicating and enjoying the process.

Keywords: pair programming, collaborative learning, MIS curriculum, collaborative programming

Introduction

Pair programming is a collaborative programming practice that has been studied often with computer science students and professional programmers (Nagappan et al., 2003; Salleh, Mendes, & Grundy, 2011; VanDeGrift, 2004; Woszczynski, Guthrie & Shade, 2005). Pair programming places participants in dyads, working in tandem at one computer to complete programming assignments. Each student takes on one of two roles, the "driver" or the "navigator." The "driver" controls the mouse and keyboard while the "navigator" makes suggestions, points out errors, and asks questions. The partners must routinely switch roles in order to gain the benefits of each role (NCWIT, 2009). Pair programming is a learning strategy derived from cooperative learning theory where instructors use cooperative methods to teach students various subjects (Slavin, 1999). While students

Editor: Keith A. Willoughby
Submitted: July 29, 2016; Revised: October 30, November 14, 2016; Accepted: December 13, 2016

learn in teams and achieve group goals, they are assessed individually. The terms “pair programming” and “collaborative programming” are often used interchangeably (Cockburn & Williams, 2001).

Although a wealth of prior literature discusses pair programming for Computer Science (CS) majors, very few studies have focused on pair programming for students majoring in Management Information Systems (MIS). The contribution of this research will be to fill the gap in the area of pair programming research as it relates specifically to MIS majors. Prior research by Woszczynski et al. (2005) found that, since MIS is a less technical degree, students often struggle in the programming course that is a part of the MIS curriculum. They suggest IS educators should explore every method that may improve success rates (Woszczynski et al., 2005). One of the researchers in this research study observed many MIS students struggling to develop programming skills in the introductory programming course, and the researchers were curious to determine if pair programming would improve student learning. Thus, the importance of this research was to determine if pair programming can benefit MIS students.

A review of 73 pair programming studies identified several benefits of this practice, including technical productivity (time spent on the program), improved program/design quality, better academic performance, and greater perceived satisfaction (Salleh et al. 2011). Pair programming improved retention within the CS major, which was defined as students who were more likely to pursue a higher level programming course and/or eventually to pursue a degree in CS (McDowell, Werner, Bullock, & Fernald, 2006). Although there are benefits to using this practice, there are several potential disadvantages, such as partners who do not get along or do not work well together. In the business world, the initial introduction of pair programming effectively doubles the time spent on a project; however, subsequent application of the practice results in only about 15% more time being spent on a project (Cockburn & Williams, 2001). Moreover, the resulting robustness of a pair programming project leads to reduced costs in correcting defects because errors are discovered earlier in the development process. Cost savings resulting from early detection of errors far outweigh the increased programming time (Cockburn & Williams, 2001. Still, a firm must closely examine the cost/benefit aspect of the pair programming practice.

Proficiency in advanced programming concepts was identified as a likely outcome in a pair programming experiment with CS majors. The experiment resulted in “a laboratory environment conducive to more advanced, active learning.” (Nagappan et al., 2003, p. 3). The study found that lab time was more productive and less frustrating than individual work (Nagappan et al., 2003).

The main goal of this study was to determine if pair programming could enhance the software development abilities (i.e., solid knowledge of the programming language) of MIS students by making them more proficient in advanced programming concepts upon completion of one course.

The research question was: *What are the particular benefits of pair programming in the MIS curriculum?* The benefits to be explored were based on four broad categories identified by Salleh et al. (2011) in their research: technical productivity, program/design quality, academic performance, and perceived satisfaction. MIS undergraduate students in the required introductory software design course were the participants in the experiment, which was conducted at a large university in the southeast United States. After a literature review, this paper next describes the research methodology. Results are presented and discussed, followed by conclusions and study limitations.

Literature Review

Research supports collaborative learning (i.e., pair programming) as preparation for work as an IS professional (Taneja, 2014). IS professionals are often the liaison between technical and non-technical members of an organization working together on project teams. As such, “It is important for instructors to enable students to experience teamwork and collaboration while preparing them for their professional careers" (Taneja, 2014, p. 181). The IS 2010 Curriculum Guidelines for Undergraduate Degree Programs in Information Systems include the need for collaborative learning in an MIS curriculum (Topi et al., 2010). A study focused on project-based team learning, using a group of MIS students, found that it is essential to create an environment for active participation and a collaboration mode in order for collaborative learning to be successful (Andres & Shipps, 2010).

In one study positive "pair pressure" produced better quality programs and allowed students to learn new languages faster compared to individual programming. There were fewer defects in the programs, with students reporting that defect removal was easier with lower frustration levels when debugging compared to individual programming. In the study, students reported satisfaction in jointly translating the customer requirements into product design and also reported that they were more productive and motivated when they worked with a partner because they kept each other focused on the task at hand (Williams & Kessler, 2000). Another experiment identified continuous reviews that lead to fewer defects in a finished program as a key benefit in pair programming. When a programmer works with a partner, there is always someone inspecting the code and, therefore, defects are identified early on (Cockburn & Williams, 2001).

Inattentional blindness occurs when the programmer is so focused on the task of coding that errors are missed that would otherwise have been caught. Reduction or elimination of inattentional blindness was a benefit in one study (Wray, 2010). The navigator catches these errors, thus reducing inattentional blindness and improving program quality. An identified disadvantage, however, is the risk of pair fatigue when programmers work together closely for an extended period of time. Eventually, the pair starts to miss the same things, therefore losing the benefit of two sets of eyes (Wray, 2010). In his own experience with pair programming, developer Stuart Wray (2010) chatted with his partners, and they reminded each other of things to be done. Another study showed that most students in an introductory programming class liked working in pairs, thought it improved their grade, and helped them work better with each other (Cliburn, 2003). Yet another study led researchers to believe that pair programming helped prepare to review and build upon existing code (Goel & Kathuria, 2010).

A similar research study on pair programming explored the retention impacts of this practice (McDowell et al., 2006). Students who programmed in pairs were more likely to pursue a higher level programming course and/or eventually to pursue a degree in CS. In the experiment, 554 students completed programming assignments. Some students were paired while others programmed individually. Within a year, 84.9% of the paired students, compared to 66.7% of the non-paired students, had enrolled in a higher level programming course. The paired students enrolled in the higher level course were more likely to succeed in their first attempt compared to the non-pairs, 65.5% versus 40%. These findings support pair programming as a strong retention practice.

Pair programming benefits also extend outside the classroom (Cockburn & Williams, 2001; Muller, 2006). Using interviews and controlled experiments to show improved organizational effectiveness due to collaborative programming, Cockburn and Williams (2001) investigated these benefits. They found that pair programming increases job satisfaction because people in pairs have a more pleasant time doing their jobs while working with a partner. Students in this research study survey gave responses similar to "It was a good exercise, I enjoy working with a partner."

There are potentially some costs associated with programming in pairs. In an educational environment, pair programming raises concerns about the accurate assessment of an individual's ability when programming in pairs. Some students in a pair may receive higher scores or undeserved credit for successfully completing a programming assignment (Hahn, Mentz, & Meyer, 2009). After noting that students typically achieved significantly higher scores for pair programs compared to individual programs, researchers Hahn, Mentz, and Meyer (2009) explored different assessment strategies that would provide a more reliable way to evaluate an individual's programming ability. In a work environment, managers may view pair programming as an inefficient use of resources with two people doing the work of one. Cockburn and Williams (2001) briefly explored some of the costs associated with pair programming and noted that expert programmers preferred to work alone because they thought they could work faster and did not have to accommodate another person. Besides being nontraditional, reluctance to share their personal code was another reason why expert programmers did not support pair programming. In this research study, the researchers observed and received feedback from some participants that pointed to personal conflict between pairs causing an unsuccessful pair programming relationship.

Methodology

The researchers developed a lab experiment to answer the research question. This method was consistent with previous pair programming studies (Mok, 2014; Nagappan et al., 2003; Salleh et al., 2011; VanDeGrift, 2004). The Salleh meta-analysis found that the most popular research methodology across 73 studies was formal experiments, used by 59% of the studies. Only 14% of the studies used surveys alone. The researchers in this study created a survey for participants to take after completing the experiment to provide them with feedback. Some of the survey questions were taken from a validated survey provided by the National Center for Women and Information Technology (NCWIT, 2009).

Study participants were undergraduate students in a face-to-face section of a required software design and development course taught by one of the researchers. This course introduces the students to programming using *Visual Basic.Net*. To determine how CS and MIS programs differ, we searched university websites for these two majors in approximately 50 different universities, including both private and public schools all over the United States (AACSB International, 2015). CS and MIS curricula vary in the number of required programming courses, with CS majors typically taking more than MIS majors. With few exceptions, MIS students were generally required to take only one software design/programming course, while CS majors were required to take two or more. These findings indicate that CS majors have more opportunities to perfect their programming skills. Although most MIS majors will not pursue careers as programmers, they must understand enough programming to be successful in their careers. MIS majors are often involved in analysis, requirements gathering, system design, and/or management of software development projects, during which they will likely encounter programming issues. Consequently, an MIS major must have knowledge of some high-level programming concepts in order to critique and provide guidance to users.

Analysis of the results of this study was accomplished by grading the resulting programs using a rubric (see the Appendix). The researcher who taught the course served as the expert and evaluated the programming code structure and form design. The program grading rubric in the Appendix was used to analyze the form design. Code content was also analyzed by looking for certain required coding constructs and how well they were implemented. Code was also graded on the quality of program execution and the output correctness. Researcher direct observations were combined with student perceptions extracted from the survey results. This is consistent with the review of pair programming studies by Salleh et al. (2011), who found that pair programming effectiveness was generally measured using four categories: technical productivity, program/design quality, academic performance, and student satisfaction. Technical productivity was used by 44% of the studies, and program/design quality by 43%. Students' perceived satisfaction was used for subjective analysis (Salleh et al., 2011). The researchers for the present study incorporated these categories in their analysis.

The researchers conducted a pilot experiment to practice the process. They made several minor changes after the pilot to improve the actual experiment. After the pilot, they conducted two separate experiments with two different classes in two different semesters. Details about the pilot and subsequent experiments follow.

Pilot Experiment

The pilot experiment occurred during the spring 2014 semester with 11 students. At this point in the semester, half the textbook had been covered and three individual programs had been completed. Four randomly selected students picked a partner for their pair. This resulted in four pairs and three solo programmers to work on the programming experiment. All students were to complete a program from the course textbook that was not revealed until the day of the experiment. The pilot began with a brief introduction to pair programming (NCWIT, 2009), then students read the program requirements and determined how to complete them. The assignment required the use of low-level concepts such as *data conversion, simple calculations,* and *input validation*, as well as high-level concepts such as the use of *sub-procedures, combo boxes, nested Ifs,* and *splash screens.* The goal of the program was to create a Windows application to compute the yearly cost of commuting to work via two different transportation modes--train or bus (Hoisington, 2014). The instructions were in a format familiar to the students since they were from the course textbook and they explicitly indicated the purpose, processing steps, and any special considerations. There was no diagram of what the form should look like, however, so students had to design the look of the form and find a picture to put on the form. The instructor/researcher took a couple of minutes to clarify the instructions, particularly the elimination of one mode of transportation specified in the problem. Participants used their own laptops. The experiment lasted 75 minutes including instructions from the researchers. After the pilot, the researchers reviewed the outcome and changed the experiment format due to issues encountered during the pilot experiment, such as non-working laptops, method of pairing students, and student skill level at the time of the experiment. To remedy these issues, the researchers decided to provide students with computers to use during the experiment. They also made selection of partners completely random, as research shows that the random matching of pairs for pair programming was similar to real world environments where workers are occasionally asked to collaborate on programming projects (Nagappan et al. 2003).

Pair Programming Experiment 1 - Fall 2014

The first research programming experiment took place for 75 minutes near the end of fall 2014, after all concepts in the textbook had been covered, and the students had completed eight Visual Basic programs individually. A few days prior to the experiment, students read two documents that briefly explained the concept of pair programming. They also received a one-page document describing how pair programming was conducted (NCWIT, 2009) and a one-page document (Fun with Pair Programming) about the "Do's" and "Don'ts" of performing pair programming (North Carolina State University, 2008).

Twenty-one undergraduate students participated in the experiment and drew numbers on the day of the experiment that indicated their pair number or assigned them to work alone. This resulted in eight pairs and five individuals to work on the experiment. Although students had worked in teams during the semester to complete a group project, none of the experiment pairs was the same as those teams. Pairs sat at a workstation that consisted of laptop access, a table, a keyboard, and a large wall-mounted monitor. The monitor helped observations by the researchers. The experiment was also video-recorded so that researchers could review the experiment later. The solo programmers were the control group for the experiment. They sat at individual desks in the same room. All students were allowed to use their textbooks. Students and pairs were referred to by number and not name so they could remain anonymous on the video although it would be viewed only by the researchers.

The experiment began with a two-minute video recorded by one of the researchers to introduce the experiment, to summarize the details of pair programming, and to inform the participants of their right to refuse being videotaped. Students then received another copy of the Fun with Pair Programming document telling them about what to do and what not to do in their pair programming lab (North Carolina State University, 2008). Next, students were given the same written programming assignment instructions from the textbook that were used in the pilot. The pairing, assigning workstations, and introductory instructions took approximately five minutes, leaving 70 minutes for the experiment.

After deciding on their own who would assume which role first, students began the first 20-minute session. Both researchers remained in the classroom during the experiment to answer questions and to observe. The researchers called time at the end of each 20-minute segment, requiring pairs to switch roles and then begin the next session. In total, three 20-minute sessions were completed. At the end of the last session, all students submitted their work into Blackboard, the course management tool. All students completed the survey about the experiment anonymously after they left the lab. Extra credit points were awarded for completing the survey only, not for the results of the program.

Pair Programming Experiment 2 – Spring 2015

The second experiment was held at approximately the same point during the spring 2015 semester as in the previous semester. Students were prepped in the same manner as used in the fall experiment. Nine students participated in the experiment. Students were again paired randomly. This resulted in three pairs and three individuals to work on the experiment. Students sat at the same type of workstations as in the fall.

The experiment proceeded in the same manner as the previous one, but the time was increased to 135 minutes, because no individuals and few pairs finished the program in the first experiment. The pairing, finding a workstation, and researcher instructions took approximately 15 minutes, leaving 120 minutes for the experiment.

Students began the first 20-minute session, once again with each pair deciding on their own who would assume which role first. Researchers remained in the classroom, and called time after 20 minutes. When finished, students submitted their programs through Blackboard, and completed the same survey, receiving extra credit points for completing the survey only.

Results

Demographics

Populations for the first and second research studies were small, with 21 participants and 9 participants respectively, which is not uncommon for this type of experiment. Previous studies have also involved small numbers of participants for pair programming experiments (Muller, 2006). Table 1 illustrates similarities and differences in the students involved in both experiments. In the spring experiment, none of the students had any programming experience, while about a fourth of the students in the fall experiment had experience. The gender mix for both experiments was approximately one-third female and two-thirds male. Finally, students in the fall group spent a little more time studying for the course during the semester than the spring group.

Table 1. Demographics

		Fall 2014 experiment	Spring 2015 experiment
Prior Programming Experience	0 months	77%	100%
	1 - 6 months	18%	0%
	7 - 18 months	5%	0%
Students' expectation of final course grade	A	64%	11%
	B	23%	78%
	C	13%	11%
Students' college level	Freshman	4%	0%
	Sophomore	0%	0%
	Junior	23%	56%
	Senior	73%	44%
Gender	Male	71%	67%
	Female	29%	33%
Students' Age	20	5%	45%
	21	38%	33%
	22	48%	11%
	24	9%	0%
	>24	0%	11%
Average number of hours spent on homework and lab each week	Less than 1 hour	5%	0%
	1 to 3 hours	30%	56%
	4 to 6 hours	55%	22%
	6 hours or more	10%	22%

Programming Results

Expert opinion was used to determine the quality of the program design and structure, similar to 19% of the studies identified by Salleh et al. (2011). The instructor/researcher served as the expert and graded all programs from the experiment using the detailed rubric in the Appendix. The rubric analyzed the form design as well as code design and functionality. In the grading process, a program was considered completely successful if it did not crash when run and if it produced correct output results. The program also needed to be well structured, and include sub-procedures, input validation, and correct conversion of data to and from both numeric and string data types. All of these methods were taught during the course. The observations of the expert from the grading rubrics about the program, code structure, and form design quality were discussed with the other researchers and are shown in Tables 2 and 3. The researchers made observations about student behavior during the experiments and upon reviewing the recordings. The results of the two experiments were not compared because the time limits were different for each experiment and a comparison was not beneficial to the purpose of the study. Instead, we compared the pair results to the individual results of each experiment. In the fall experiment, three programs could not be graded, two from pairs and one from an individual, due to corrupted or lost programs. This resulted in a total of six pairs and four solo programmers. For the spring experiment, all programs were submitted successfully and all were graded.

Table 2 shows grading results for the fall experiment, including total score and abbreviated comments. In this group, the top two programs were created by Pair 5 and Pair 6. Only Pair 6 created a program that produced correct results without crashing; however, the validation code had a flaw, and while the processing for bus and train was correct, it was not placed in separate sub-procedures as required. The program from Pair 5 ran and produced correct results but eventually crashed due to a minor error. All other pairs and individuals created the form correctly, although there were problems in the coding that prevented the programs from running. Creating a form correctly was relatively easy, so it was only worth about 20% of the grade; hence, many of the other grades were low because there were problems in the code. Several of the programs were incomplete, and most did not use high-level concepts such as sub-procedures and processing of combo boxes or contained incomplete sub-procedures. Unfortunately, several of the solo programs also had problems with low-level concepts such as data conversions. This was surprising since several of the solo programmers were top performers throughout the course, and this basic concept was covered early in the semester. This may have occurred due to time constraints, or because some of the top performers the instructor assumed had worked alone had assistance.

All individual scores were the same as or below all pair scores. The average score for pairs was 56.83 and for individuals was lower at 44.25. Also, the only scores above 60 were pair scores. Although some individuals scored the same as or close to a pair program score, there were no individuals with a high score.

Table 2. Program Grading Results for Experiment 1 - Fall 2014

ID	Grade	Experiment 1 – Comments
Pair 1	44	Program produced results, but, did not follow requirements. Option for selecting train or bus missing. No separate sub-procedures for processing Train vs. Bus. No input validation.
Pair 2	54	Program structure had “Build” errors & would not run
Pair 3	50	Program structure very good but validation, train and bus sub-procedures missing.
Pair 4	40	Program requirements not followed and extra input was requested. Calculations missing.
Pair 5	75	Good code structure and design. Program worked but crashed on minor error
Pair 6	78	Good code structure and design. Program worked. Missing some validation. No separate sub-procedures for bus and train. A sub-procedure for validation was used.
Indv. 1	44	Program code was incomplete, sub-procedures labeled but not completed. Program crashed due to missing code.
Indv. 2	52	Structure of program was good but incomplete. Errors in data conversion. No separate sub-procedures for bus and train. Program froze and would not run.
Indv. 3	47	Program code was incomplete and incorrect. A sub procedure for train was labeled but not defined or used. No sub-procedure for bus. Some data conversions were incorrect.
Indv 4	34	Form was good but program was very incomplete. Program crashed. Missing variables, missing data conversions, no separate sub-procedures for train or bus.

Programs in the spring experiment were graded using the same rubric, and the results are shown in Table 3. In this experiment, only the program from Pair 4 ran flawlessly without error and produced correct results. The program from Individual 1 worked, but had similar issues as other low scoring pair programs as it was very simplistic and did not follow requirements. Additionally, some individuals had problems with low level concepts such as data conversion, validation, and splash screens. All pairs and individuals created the form correctly. Overall, the pairs scored higher than the individuals. The average score for pairs was 73.33 and for individuals was lower at 52.67. The extra time for this experiment proved useful. Two pairs and two individuals completed the program at the beginning of the fourth 20-minute session, only slightly more time than that for the fall experiment. One pair and one individual required the entire two-hour time frame. All individual scores were the same as or below all pair scores and only the pair scores were over 60. There were no individuals with a high score.

Table 3. Program Grading Results for Experiment 2 – Spring 2015

ID	Grade	Experiment 2 - Comments
Pair 1	69	Program runs and produces results, but is very simplistic, did not use sub-procedures. Validation process incomplete.
Pair 3	47	Program used NO higher level concepts such as combo box, Nested If or Splash Screen processing
Pair 4	104	Program was written extremely well, ran and produced correct results. It worked correctly and included extras not requested. Extra credit given for extras.
Indv 1	46	Individual didn't seem to understand requirements at all. Program was coded in most simplistic way and many low-level coding processes were missing like data conversion and validation. No Splash Screen.
Indv 2	55	Program was coded in most simplistic way with no sub-procedures. Program crashed.
Indv 3	57	Individual included outline of correct structure but did not complete all sections. Some portions of code were good. Program eventually froze. Some sub procedures were incomplete and some partially functional.

In summary, for both experiments, the programs created by the pairs were structured well and included advanced concepts such as Nested Ifs and sub-procedures, some of which were started but not completed. Two individual programs initiated sub-procedure events by labeling them but did not complete them with code. The problems with not creating proper sub-procedures were likely due to students who understood the concepts that were needed but could not follow through to completion due to insufficient knowledge or time constraints in the fall experiment. Overall, the pairs did a better job than the individuals with the high-level concepts, although several pairs were still unsuccessful. It was encouraging to see that students had some idea of what was needed when they labeled the sub-procedures. It was discouraging to see that some of the students had trouble with some low-level concepts, e.g., using correct data types to define variables and correctly converting data to and from string data types. This may have occurred because of carelessness or time constraint pressure since the students' knowledge of defining and converting variables was established early in the course. Another possibility is that students received assistance when completing programs outside the classroom. Finally, we believe the pairs were able to complete high-level concepts by combining their partial knowledge into a complete execution of high-level concepts. High-level concepts were not present in all items developed by pair programmers, but it did not happen at all with any of the solo programmers.

An ANOVA test of all the program scores for both semesters combined compared mean scores between pairs (*M* = 62.33, *SD* =20.85) and individuals (*M* = 47.86, *SD* =7.78). However, the results were not statistically significant, $F(1,14) = 6.99$, $p = .105$, *partial* $\eta^2 = .177$. An ANOVA test for each of the program constructs in the grading rubric was also conducted, such as combo-box, splash screen, and textbox validation. Only "Correct Data Conversion" with a possible point value of 10 was statistically significant, $F(1,14) = .156$, $p = .040$, *partial* $\eta^2 = .267$. Pairs had a higher mean score (*M* = 8.00, *SD* =3.12) than individuals (*M* = 4.14, *SD* =3.72). These results are likely due to the point spread of the grading rubric which was based on expert opinion.

Researchers' Observations

A fascinating aspect of the study was the researchers' observations and how these observations aligned with the students' perceptions from their questionnaire responses. The researchers observed the students during each experiment, then met to review the recordings and discuss their observations. During the fall, the researchers observed two solo students not focusing, instead looking around at what other students were doing. These two appeared slightly behind in progress compared to the pairs. Some individuals were working diligently, while others ran into problems and could not recover. In one instance, a struggling individual disengaged, making conversation with a nearby individual on unrelated topics.

Some pairs also had problems. First, getting the students to work as pairs was challenging. In the first 20-minute session, some students were driving and navigating simultaneously. Additionally, one pair started off badly and was never able to recover; a member of this pair took over the keyboard and began driving without discussion. The navigator in this pair was unhappy about that action and commented to the researcher. They did not talk to each other throughout the entire experiment, and each worked alone. This disengaged pair never worked as a cohesive unit and did not turn in a result. One member of this pair when not driving was looking around, using his cellphone, and even napping while waiting for his turn. In a different pair, one member dominated the process and was somewhat condescending to the partner. However, this pair continued to make progress. All other pairs appeared to be working well.

During the second 20-minute session, things began to improve. Several pairs were being very communicative and progressing well. One pair seemed to really enjoy the process and were working together, communicating well, laughing, and having fun. Three of the individuals appeared to be progressing and were coding, while two were struggling with the assignment. During the final 20-minute session, two pairs completed the program. All remaining pairs were coding and most were problem-solving together at this point. The disengaged pair continued to take turns working individually. Overall, it was encouraging to observe the pairs enjoy the experience, become engaged, actively work together, and exchange ideas. This is especially refreshing since the instructor/researcher had previously observed some of these same students being stressed while trying to program individually during the semester. Our observations were similar to those identified by Nagappan et al. (2003) in their research where they also found labs with pair programming were more vocal and interactive than solo programming labs.

In the spring, all three pairs appeared to work well together. One pair seemed to really enjoy the process; both students were smiling throughout the experiment. Another pair struggled throughout the experiment, became very frustrated but continuously communicated with each other, going back and forth about what was really required. They even commented that they were the wrong two people to be paired together. Although it took them the entire two-hour session, they never stopped communicating, and although things were tense at one point, they were respectful to each other and got through it. The individuals all appeared to struggle, some frantically flipping through the pages of their textbook and appearing to try multiple methods during the coding of the program. These individuals eventually finished, but their programs overall were not successful.

Student Perceptions

In fall 2015 and spring 2016 the pair programming concept was implemented as a part of the Software Design and Development course curriculum. Of the 8 programs assigned to the students, half were individual and half were pair programs. The results of the student surveys from the two experiments and the two semesters where it was incorporated in the course are listed below. Table 4 shows results of the students' perceptions about their partners. Although researcher observations are important, student perceptions are also important. Overall, the results are in favor of pair programming because the students perceived that the results and the experience were better, which is consistent with results found in studies of CS/SE students (Salleh et al., 2011).

Table 4. Survey Question Results - Student Abilities

Survey Question	Fall 2014 Averages	Spring 2015 Averages	Fall 2015 Averages	Spring 2016 Averages
My partner's technical competency is 1. Much better than mine 2. Somewhat better than mine 3. About the same as mine 4. Somewhat less than mine 5. Much less than mine	2.95	3.14	3.00	3.09
My partner and I were: 1. Very compatible 2. Somewhat compatible 3. Not at all compatible	1.25	1.17	1.33	1.55
How did having a partner affect the quality of your programs compared to what you would have written alone? The programs were: 1. Lower in quality 2. About the same 3. Higher in quality	2.32	2.57	2.13	2.70

Table 4 shows survey questions designed to evaluate the students' perception of their programming abilities and the overall quality of a program written by a pair. First, students in both experiments felt their technical abilities and that of their partners were about the same, with an average around 3. The question about partner compatibility is important since pairs were randomly combined and compatibility could have an effect on the result of the pair programming experience and the final result. The average for both experiments was very close to 1, indicating pairs were "very compatible." The researchers' observations were similar to the student's perceptions when analyzing compatibility. Finally, the students were asked about the quality of their pair program compared to the quality of their expected result if they had produced the program alone. In this case, the fall group average was 2.32, while the spring group average of 2.57 indicated that students perceived the result was between "about the same" and "higher quality."

Table 5 shows the results of student perceptions of pair programming from the survey. The first nine questions in the table are about student perception of the partner experience. The averages of the fall group overall were lower than those of the spring group; however, most students agreed that the pair experience was beneficial and helped them program. The main difference between the groups was on the question: "It was easy for me to get my pair programming partner to answer my questions." The fall group average response was 3.10, which indicates "agree." The spring average response was 3.71, which is closer to "strongly agree." The researchers did not have an explanation for this. Some possible explanations are differences in the students themselves, a smaller class size in the spring, or more experienced researchers. The majority of participants felt the pair programming process was beneficial and made them more confident about their code, enhanced problem-solving, and clarified unclear concepts. The question about wanting to change partners produced interesting results. The average score for both groups was very close (1.75 and 1.86), which indicated overall satisfaction with their partners. This was surprising as two pairs in the fall were very dysfunctional.

Table 5. Survey Question Results – Student Perceptions

SURVEY QUESTION Strongly Disagree = 1; Disagree = 2; Agree = 3; Strongly Agree = 4	Fall 2014 Averages	Spring 2015 Averages	Fall 2015 Averages	Spring 2016 Averages
Having a partner made it easier to complete assignments.	2.90	3.14	2.88	2.91
Having a partner made me feel more confident that the code was reliable.	2.95	3.14	2.88	2.91
It was helpful to discuss programming problems & solutions with a partner.	3.10	3.43	3.25	3.27
Having a partner is beneficial for learning to read another programmer's code.	3.10	3.29	3.0	3.18
It was easy for me to get my pair programming partner to answer my questions.	3.10	3.71	2.88	2.91
My partner and I were equally matched in terms of the pace at which we solved programming assignments.	2.85	3.0	2.63	2.82
I wanted to change to a different pair programming partner.	1.75	1.86	1.88	1.82
I am comfortable being the "driver"	3.29	3.25	3.5	3.27
I am comfortable being the "navigator"	3.15	3.33	3.38	3.27
Pair programming leads to more success than individual programming.	3.15	3.5	2.75	2.82
Pair programming should be part of every class that requires programming assignments.	3.0	3.63	2.5	2.91
I would recommend pair programming to other students.	3.2	3.5	2.63	3.09

The last three questions in Table 5 reveal the students feelings about using the pair programming practice. These questions measured student perception of pair programming as a pedagogical tool for the course, including whether they would recommend this practice to other students. The average scores here all ranged between "agree" and "strongly agree." In fact, the spring group averaged 3.63, which was closer to "strongly agree" when asked if pair programming should be a part of every programming class. Other research also found students to be positive about working in pairs in the future after their experiment (Nagappan et al., 2003). This result, as well as other results, convinced the researchers to incorporate pair programming into the curriculum of the fall 2015 course.

Overall, the average responses were more positive in the two experiments than in the course implementation. In the experiments the average response was 3.0 and higher (3=Agree) when students were asked about the success of pair programming for learning programming concepts. In the course implementations, the results were as low as 2.5 (between disagree and agree) and only one average was over 3 for the same three questions. The results for the course implementation are likely different because pair programs included in the course had an impact on the students' grades where there was no grade impact during the experiments. However, the instructor felt the results were positive enough to continue to include pair programming in the course.

The instructor observations during the implementation for semesters fall 2015 and spring 2016 were different. During the fall 2015 semester pair programming allowed the instructor to interact more with the students while they were programming and students asked more questions about their programs. However, unlike previous classes, the new implementation allowed students to work on programming assignments during in-class labs instead of outside of the classroom. During the spring 2016 semester there were problems with "loafers" in some pairs. The instructor had to intervene to manage this issue. Although this was disappointing, the pair programming practice has had enough value in the course to continue using it in future semesters.

There were some open-ended questions in the survey to capture student perceptions in their own words. Some of those perceptions were:

> "I thought it was a good exercise, I enjoy working with a partner."
>
> "I wish I had more time to use this method. Possibly during guided development or certain chapter programs."
>
> "I liked the process overall, I just wish that I took it more seriously in the beginning."

Finally, when asked what they would change, one student indicated the following, which was encouraging: "working on a program that was more difficult."

Discussion

The significance of the findings in this study with MIS majors is their similarity to results for CS majors. While the similarity of the findings for the two types of majors may not seem remarkable, it is in fact very important. The results point to an area in the MIS curriculum that can be improved to produce more confident students who are prepared to work as collaborating member of a pair in the workplace.

As an instructor of a MIS introductory software design course for many years, the lead researcher has on several occasions encountered students who, although they excel in their other courses, are stumped by programming. The resulting anxiety for these students causes them to contemplate changing majors, which directly impacts retention. If a CS major cannot program it can be difficult for them to find a job in their field or even to graduate in their major. However, the majority of careers for MIS majors will not require highly technical tasks such as advanced programming. Instead, they will need to understand enough about software development to act as a liaison between the customer and the technical staff. In other words, it is likely a MIS major can survive even if programming is difficult for them. Therefore, it would be wise to structure a software design and development course in a way that allows MIS majors to learn and use the concepts while collaborating with a partner since that is more likely to produce positive results. Survey results indicated that the majority of participants in these experiments felt the pair programming process was beneficial and made them more confident about their code, enhanced problem-solving, and clarified unclear concepts. One of the researchers has observed several MIS students who struggled to pass the programming course eventually land good jobs and go on to have very successful MIS careers. Although this is not statistically documented in our findings, it is important to the usefulness of our results.

Future research should further explore pair programming in the MIS curriculum to determine if this technique can improve retention of MIS majors or reveal other specific benefits. As noted, the researchers continued this research by altering the software design course in the fall 2015 and spring 2016 semesters to include pair programming, and those results were shared in the Student Perceptions section. The intention is to continue using pair programming in future semesters.

Conclusion

The researchers concluded that the benefits of pair programming with their MIS students were very similar to those found in CS/SE studies. Students in pairs appeared to enjoy the process of creating the program much more than those who programmed individually. This was consistent with research that hypothesized that students in paired labs would have a positive attitude towards working in collaborative software development environments. The same research also found pair programming to be beneficial for non-CS majors (this could include MIS), but not necessarily for CS majors (Nagappan et al., 2003).

The researchers concluded, from their observations and student survey responses, that students in pairs shared ideas that helped them learn from each other. Sharing ideas and learning from a partner have previously been identified as benefits of pair programming (Muller, 2006; Salleh et al., 2011). The goal of our study was to determine if pair programming would allow MIS students to become more proficient after completing one software course in four broad categories: technical productivity, program/design quality, academic performance, and perceived satisfaction. Our results indicate academic performance in pairs was better; overall the pairs in both experiments performed better than the individuals even though some of the individuals were "A" students in the course. Technical productivity and program/design quality was generally better with the pairs. Only pairs in these experiments were able to complete the assignment successfully using high-level concepts such as sub-procedures and producing correct results without crashing the program. In both experiments we observed students who had not worked together previously but collaborated well during the experiment. We even had some pairs in both experiments that were laughing and enjoying themselves while they worked. We hope this type of pair programming experience will prepare the students for working collaboratively in the workplace where pair programming is being used as a part of extreme programming practices (Muller, 2006). Our results for MIS majors were consistent with results from the study with CS and non-CS majors by Nagappan et al. (2003) where their results showed "paired and solo programmers have comparable scores in the projects though in some cases paired programmers have marginally higher scores than the solo students."

There were some limitations to this study. First was the time constraint of 75 minutes; this appeared to hinder completion of more high level concepts like sub-procedures. After grading the programs from the fall experiment and noting that more than half the students indicated that the time limit was too short (Table 3), in the spring the time was extended to 120 minutes. In spring, two of the groups and two of the individuals finished between 70 and 82 minutes, with one group and one individual requiring the entire 120 minutes. It appears although the 75 minute time frame was perceived as a limitation, in fact several students were able to finish within that time limit when given more time. Thus, it remains uncertain what the optimal time limit would be. Another limitation common to surveys was the self-reporting of students on their own behavior. The small sample size, although common in pair programming experiments, may make it difficult to generalize these results.

References

AACSB International. (2015, April). Retrieved from AACSB International website: www.aacsb.edu

Andres, H. P., & Shipps, B. P. (2010). Team learning in technology-mediated distributed teams. *Journal of Information Systems Education, 21*(2), 213-221.

Cliburn, D. C. (2003, October). Experiences with pair programming at a small college. *Journal of Computing Sciences in Colleges, 19*(1), 20-29.

Cockburn, A., & Williams, L. (2001). The cost and benefits of pair programming. In G. Succi & M. Marchesi (Eds.), *Extreme programming examined* (pp. 223-243). Boston: Addison-Wesley Longman Publishing Co.

Goel, S., & Kathuria, V. (2010). A novel approach for collaborative pair programming. *Journal of Information Technology Education, 9*, 183-196. Retrieved from https://www.informingscience.org/Publications/1290

Hahn, H. J., Mentz, E., & Meyer, L. (2009). Assessment strategies for pair programming. *Journal of Information Technology Education, 8*, 273-284. Retrieved from https://www.informingscience.org/Publications/694

Hoisington C. (2014). *Microsoft Visual Basic 2012 for Windows, Web, Office, and Database Applications: Comprehensive.* Stamford, CT: Course Technology, Cengage Learning.

McDowell, C., Werner, L., Bullock, H. E & Fernald, J. (2006). Pair programming improves student retention, confidence, and program quality. *Communications of the ACM*, *49*(8), 90-95.

Mok, H. (2014). The flipped classroom. *Journal of Information Systems Education, 25*(1), 7-11.

Muller, M. (2006). A preliminary study on the impact of a pair design phase on pair programming and solo programming. *Information and Software Technology*, *48*, 335-344.

Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C., & Balik, S. (2003). Improving the CS1 experience with pair programming. *Proceedings of the 34th SIGCSE: Technical Symposium on Computer Science Education.* 359-362. Reno, NV: ACM Special Interest Group on Computer Science Education.

NCWIT. (2009). *Pair programming-in-a-box: The power of collaborative learning*. Retrieved November 2013, from National Center for Women & Information Technology: www.ncwit.org/pairprogramming

North Carolina State University. (2008). *Fun with pair programming!* Retrieved November 2013 from http://www.realsearchgroup.org/pairlearning/worksheet.pdf

Salleh, N., Mendes, E., & Grundy, J. (2011). Empirical studies of pair programming for CS/SE teaching in higher education: A systematic literature review. *IEEE Transactions on Software Engineering*, *37*(4), 509-525.

Slavin, R. E. (1999). Comprehensive approaches to cooperative learning. *Theory into Practice, 38*(2), 74-79.

Taneja, A. (2014). Enhancing student engagement: A group case study approach. *Journal of Information Systems Education, 25*(3), 181-187.

Topi, H., Valarich, J. S., Wright, R. T., Kaiser, K. M., Nunamaker Jr., J. F., Siplor, J. C., & de Vreede, G. J. (2010). IS 2010: Curriculum guidelines for undergraduate degree programs in information systems. *Communications of the Association for Information Systems*, *26*(1), 18, 359-428.

VanDeGrift, T. (2004). Coupling pair programming and writing: learning about students' perceptions and processes. *Proceedings of the 35th SIGCSE: Technical symposium on Computer science education,* 2-6. Norfolk, VA: ACM SIGCSE - Special Interest Group on Computer Science Education.

Williams, L. A., & Kessler, R. R. (2000). The effects of 'pair-pressure' and 'pair-learning' on software engineering education. *Proceedings of the 13th Conference on Software Engineering Education & Training*, Austin, TX, 59-65.

Woszczynski, A. B., Guthrie, T. C., & Shade, S. (2005). Personality and programming. *Journal of Information Systems Education, 16*(3), 293-299.

Wray, S. (2010). How pair programming really works. *IEEE Software*, *January/February*, 50-55.

Appendix

Pair Programming Exercise Grading Rubric			
Semester: **Group/Individual:**			Chapter 7 - Calculate Your Commute (no Car) Exercise
FORM DESIGN	Possible Points	Points Deduction	Comments
Splash Screen (extra credit)	0	0	
Title	2	0	
Picture selected from Web	2	0	
Combobox to select mode	5	0	
Objects labeled correctly	2	0	
Round Trip fare labeled textbox	2	0	
Days worked/month labeled textbox	2	0	
Calculate button	2	0	
Clear button (not specifically required but needed)	2	0	
PROGRAMMING CODE			
CBO_SelectedIndex Event - Objects appear after mode selected	10	0	
Intro Comments & Program Comments	3	0	
Load Event - Splash Screen timing (extra credit)	0	0	
Textbox Validation Events - positive number	10	0	
Train Processing - used sub procedures (10)	10	0	
Bus Processing - used sub procedures (10)	10	0	
Correct data types	5	0	
Correct Conversion	10	0	
Program runs	15	0	
Calculation is correct	8	0	
	100	**100**	
OVERALL COMMENTS			

Biographies

Tendai A. Dongo was an instructor in the MIS department in the College of Business at East Carolina University. Upon completion of her MBA at ECU, she joined the Graduate Programs team in the college as an Assistant Director. She has published research in the AMCIS proceedings. Her interests include pedagogical research in management information systems, leadership and organizational behavior.

April H. Reed is an Associate Professor in the College of Business, MIS department at East Carolina University. She conducts research in the area of IS/IT project management and pair programming. She is a PMI certified Project Management Professional (PMP) and has held several industry positions including Systems Analyst and Project Manager. She has published several papers on the topic of risk and virtual software development project teams in IS journals such as International Journal of Information Technology Project Management, Journal of Computer Information Systems, International Journal of Project Management, Informing Sciences and Journal of Information Technology Management. She holds a PhD in Computer Science/Information Systems from DePaul University.

Margaret (Maggie) O’Hara has been teaching Management Information Systems (MIS) since 1992. She received her PhD from the University of Georgia. At East Carolina University from 1999 – 2012, she taught a variety of graduate and undergraduate MIS subjects, in both face-to-face and online formats. From 2012 – 2015 O’Hara served as Director of Online Learning for the University of North Carolina system. In 2014, she received the United States Distance Learning Association Award for Outstanding Leadership in Distance Education. In spring 2015, O’Hara resumed her position as a full-time faculty member at ECU. She currently teaches MIS and a Leadership and Professional Development. She has published research in both MIS and Education journals and has consulted internationally on technology-enhanced teaching and learning.