Enhancing Software Development in the MIS Curriculum using Pair Programming

Full Paper

Tendai Ndabvonga-Dongo
East Carolina University
College of Business – MIS
ndabvongat@ecu.edu

April H. Reed
East Carolina University
College of Business – MIS
reeda@ecu.edu

Abstract

Management Information Systems (MIS) majors often must perfect their programming skills with one course which can be a daunting task. In an effort to enhance the software development abilities of MIS majors a pair programming lab experiment was conducted in an introductory software development course to determine if that technique would produce benefits for the MIS curriculum. Pair programming experiments are often performed with Computer Science majors but rarely with MIS majors. The researchers' observations as well as participant's responses to a survey questionnaire were analyzed after the experiment. The results indicated that pair programming may be beneficial as a pedagogical tool to a MIS students' ability to create programs using high-level concepts. Additionally, researcher observations revealed pairs worked collaboratively to produce the program while actively communicating and enjoying the process.

Keywords

Pair programming, collaborative learning, MIS curriculum

Introduction

Prior literature shows pair programming is a collaborative programming method that has been studied often in recent years using computer science students and professional programmers (Salleh, Mendes & Grundy, 2011). Pair programming is designed to allow students to complete programming assignments in small groups of two instead of individually as solo programmers. Salleh, et al. (2011) in their review of 73 quality pair programming studies identified several benefits such as technical productivity, program/design quality, academic performance and satisfaction. Pair programming has also been shown to help with retention within the computer science major (McDowell, Werner, Bullock & Fernald, 2006). During pair programming, students work in tandem at one computer while completing regular 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). Although there are benefits to using the Pair Programming approach there are several potential disadvantages, such as partners that don't get along or don't work well together as witnessed in this research study.

Although a wealth of prior literature discusses pair programming in the Computer Science (CS) major very few studies focus on pair programming in the Management Information Systems (MIS) major, therefore, it is important to investigate benefits for MIS majors. The contribution of this research is to reveal benefits of pair programming as a pedagogical tool for MIS students and specifically the potential for enhancing learning in the MIS curriculum. Unlike some other studies which investigate the quality benefits (Muller, 2006), the main goal of this study was to find a method to enhance the software development abilities of students by making them more proficient in advanced programming concepts

upon completing one course. These abilities were thought to benefit our MIS majors in their future careers in positions such as Systems Analysts.

The research question is: *What are the benefits of pair programming in the MIS curriculum?* The research was conducted using a programming experiment at a large public university in the southeast United States. MIS undergraduate students from a required introductory software design course were the participants.

Background

This section will discuss a few prior studies on pair programming and Cooperative Learning Theory which supports the effectiveness of pair programming. To determine differences in programming requirements for CS and MIS programs we conducted a search via the university websites of these two majors in approximately 50 different universities, including both private and public schools all over the United States (AACSB International, 2015). In general, the CS and MIS curriculums vary in the number of required programming courses with CS majors taking more programming courses than MIS majors. Overall, MIS students were generally required to take one software design/programming course while CS majors were required to take two or more. Of course there were exceptions where some MIS majors were required to take two. Given these findings we can deduce CS majors have the opportunity to perfect their software development skills over more courses than MIS majors. We must also acknowledge the fact that CS majors may be expected to learn more high-level programming skills than MIS majors since some will embark on careers in programming. We have encountered college students who are serious "techies" and want nothing more than to interact with a computer, they often become CS or SE majors. Although this can also be true for MIS majors, it is less often the case and we have encountered MIS majors who are worried they will be asked to program sometime in their careers. However, MIS majors must understand enough programming to be successful in their careers. For example, even if a MIS major does not have a career in software development they are generally involved in the analysis, requirements gathering and/or managing of software development projects. Consequently, a MIS major must have knowledge of some high-level programming concepts in order to be effective. Since pair programming has obvious benefits for CS majors we wanted to explore the benefits for MIS majors and focus on how pair programming would enhance the MIS curriculum.

Prior research on pair programming has identified several benefits for CS students as well as professional programmers (Cockburn, et al., 2001; McDowell, et al., 2006; Muller, 2006; Salleh, et al., 2011; Williams, et al., 2000). In their research Williams and Kessler (2000) found positive "pair pressure" produced better quality programs and allowed students to learn new languages faster compared to individual programming. In a Collaborative Development of Active Server Pages course, 20 students, were assigned a partner for the entire semester. Students were advised to work on each piece of a project as a team and not to simply integrate individually programmed work for submission. Students were provided 3 different ways to share feedback; web-based journals, anonymous surveys and a letter advising future pair programmers of the course. The results of this experiment showed that the average grade on each project was at least "A" grade work. There were fewer defects in the programs, with students reporting that defect removal was easier with lower frustration levels of debugging compared to individual programming (Williams & Kessler, 2000). Another pair of researchers, Cockburn and Williams (2001) in a separate experiment attributed continuous reviews as a key factor in pair programming that leads to fewer defects in a finished program. When a programmer works with a partner, there is always someone inspecting their code and therefore defects are identified early on. In addition to better quality code, pair programs are reviewed informally, without the feeling of anxiety associated with a formal review (Cockburn & Williams, 2001). Additionally, pair programming improves program quality because another person is looking at the code other than the programmer themselves thus reducing or eliminating inattentional blindness (Wray, 2010). Inattentional blindness in pair programming occurs when the driver is so focused on the task of coding that they miss an error that would otherwise have been caught. The role of the navigator is to catch the errors and thus reducing inattentional blindness and improving program quality. A disadvantage 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).

Pair programming benefits extend outside of the classroom (Cockburn, et al., 2001; Muller, 2006). Cockburn and Williams (2001) investigated these benefits using interviews and controlled experiments to show improved organizational effectiveness due to collaborative programming. They found pair programming increases job satisfaction as people in pairs have a more pleasant time doing their jobs while working with a partner. Similar responses were given by students in this research experiment, they indicated "It was a good exercise, I enjoy working with a partner." Similar experiences were reported by pair programmers surveyed at the University of Utah stating "it's nice to celebrate with somebody when something works" (Cockburn & Williams, 2001). In the pair programming experiment conducted by Williams and Kessler (2000), students reported satisfaction in jointly translating the customer requirements into product design. Students 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). Pair programming involves more than just two programmers taking turns to develop a program. In order for the technique to be effective, it must be a very interactive process where the pair is constantly communicating, writing down ideas and pointing to the screen. 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.

McDowell, et al. (2006) conducted similar research on pair programming and explored the retention impacts of this method. They found that collaborative programming increased the quality of programs in terms of functionality and readability. Additionally, students who programmed in pairs were more likely to pursue a higher level programming course, and/or eventually pursue a degree in Computer Science (McDowell et al, 2006). In their experiment 554 students taking an introductory programming course at University of California, Santa Cruz were required to complete programming assignments. Some students were paired, while others were instructed to program individually. The investigators found students who programmed in pairs were significantly more likely to remain enrolled in the course compared to their non-paired counterparts. Within a year, 84.9% of the paired students compared to 66.7 % of the non-paired students who were intending to major in computer science had enrolled in a higher level programming course (McDowell et al, 2006). Those students who enrolled in the higher level course and had worked as a pair in the introductory 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 method (McDowell, et al., 2006).

Pair programming is a learning strategy that is derived from cooperative learning theory (Slavin, 1999). In cooperative learning, instructors use cooperative methods to teach students various subjects. Students learn in teams and create group goals, but they are typically assessed individually. In a comprehensive study of cooperative learning in an elementary school, Slavin (1999), discovered that teachers reported students were reading better and were more involved in class activities. In collaborative learning, students are allowed to learn programs and other course materials as a team, preferably a pair. As the students get to know each other, they realize their partner's strength and tend to become more efficient as a pair. When one partner does not understand a concept, usually the other partner does and can help explain it. Williams and Kessler (2001) observed that adopting collaborative learning through-out the semester encouraged students to view group projects as "ours" versus "my part and your part". The joint ownership creates an environment where students are comfortable to constructively criticize and recommend modifications to their partners work (Williams & Kessler, 2000).

METHODOLOGY

The researchers in this study wanted to investigate potential benefits of pair programming for teaching MIS students programming concepts. The method chosen to explore this topic was a programming lab experiment followed by a questionnaire with undergraduate students in a face-to-face section of a software design and development course taught by one of the researchers. In this course, students are introduced to programming using Visual Basic.Net 2012. The researchers began by creating a survey questionnaire for the students to take after they completed the lab experiment. The researchers decided to videotape the students to enable making observations about student behavior during the experiment. The survey was created from a validated survey provided by the National Center for Women and

Information Technology (NCWIT). Although the experiment would have been sufficient, a questionnaire was added to provide the researchers with feedback from the students. The researchers conducted a pilot experiment first in order to practice the methods for the live experiment and learn from them. This proved to be a wise decision since several minor changes were made after the pilot which enhanced the actual experiment.

This research study used a methodology that was consistent with previous pair programming studies as indicated in a recent article which reviewed 73 past pair programming studies from 1999 to 2007 (Salleh, et al., 2011). Salleh, et al. (2011) found the most popular research methodology was formal experiments which was used by 59% of the studies. Also, 14% of the studies used surveys, however, there was no mention of recording of the experiments. We felt the recording aspect was an added benefit to our research methodology and were very careful to keep the recording anonymous. As indicated in the introduction, MIS students were studied because prior literature focused mainly on CS/SE students and MIS students often take fewer programming courses than their CS/SE counterparts.

Pilot Experiment

The pilot experiment was held during the Spring 2014 semester with 11 students. It was held at a time when half of the textbook had been covered and students had already turned in three individual programs for grades in the course. Four students were randomly selected and asked to pick a partner for their pair. This resulted in four pairs and three solo programmers to work on the programming experiment. All students in the experiment were assigned to complete a program from the textbook which was not revealed until the day of the experiment. They were given a brief introduction to pair programming and then told to read the program requirements and determine how to complete it in their pairs or individually, according to how they had been assigned (Hoisington, 2014; NCWIT, 2009). Participants were asked to use their own laptops and to work in pairs and some as individuals. The entire experiment lasted 75 minutes including instructions from the researchers. After the pilot the researchers met to review the outcome and made several changes 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. It was determined students would be provided with computers during the actual experiment and to use random selection of partners.

Experiment 1

The research programming experiment was held in the Fall 2014 semester for 75 minutes at the end of the semester after all concepts in the textbook had been covered and students had completed 8 Visual Basic programs individually on their own. Students were provided two documents a few days prior to the day of the experiment that briefly explained the concept of pair programming. They were given a 1 page document describing how pair programming was conducted (NCWIT, 2009) and a 1 page document about the Do's and Don'ts of performing pair programing (North Carolina State University, 2008).

21 undergraduate students participated in the experiment and were grouped on the day of the experiment by having them select a random number which told them which pair they belonged to or if they were to work alone. This resulted in eight pairs and five individuals to work on the experiment. Pairs worked at a station which consisted of laptop access, a table, a keyboard and a large wall mounted monitor. Solo programmers used their own computers at individual desks in the same room. All students were allowed to use their textbook and had been told to bring them to class. Students and pairs were referred to by number and not by names so they could remain anonymous on the video.

The experiment began by playing a two minute video recorded by one of the researchers which summarized the explanation of pair programming. Students had also been given a written form a few days prior to the experiment (NCWIT, 2009). Students were then given a copy of the 1 page Pair Programming Do's and Don'ts document (North Carolina State University, 2008). Next, students were given written programming assignment instructions which came from chapter 7 in their course textbook (Hoisington, 2014). The programming assignment was of medium difficulty according to the textbook measuring system and was called "Calculate Your Commute". The goal of the program was to create a Windows application that would compute the yearly cost of commuting to work via train or bus (car

commuting was removed from the instructions due to the time constraints). The instructions were explicit, indicating the purpose, processing and any special considerations, however, there was no diagram of what the form should look like. The course instructor took a couple of minutes to make a few comments about the program instructions to ensure clarity. The pairing, finding a station and instructions took approximately ten minutes.

Using a timer, the students were told to begin the first 20 minute session. Each pair decided on their own who would assume which role first. Both researchers remained in the classroom during the experiment to answer questions if needed. Time was called at the end of the 20 minutes which allowed pairs to switch roles then begin the next session. In total, three 20 minute sessions were completed. At the end of the last session all students were asked to submit the programming results via Blackboard, the course management tool. Students were asked to complete a survey questionnaire about the programming experiment on their own once they left the lab. Extra credit points were awarded for completing the survey only, not for the results of the program.

.

Results

The population for this research study was small with 21 participants, however, this is not uncommon for this type of experiment. Previous studies also used small numbers of participants for a pair programming experiment such as Muller (2006) who conducted an experiment comparing pair programming with solo programming using 18 students. In this same article Muller (2006) cited two other pair programming studies using an experiment with a small number of participants, Nawrocki and Wojciechowski in 2001 with 21 CS students and Nosek in 1998 with 15 professional programmers (Muller, 2006).

The demographics (see Table 1) shows the majority of the students were male (71%) and seniors (73%). Most had no prior programming experience prior to the class (77%). Finally, most were hard working students with more than half spending an average of 4 to 6 hours per week on homework and labs for the course and three quarters of the class expecting a final grade of A or B.

Prior Programming Experience	0 months	77%
	1 - 6 months	18%
	7 - 18 months	5%
Student's expectation of final course grade	A	64%
	B	23%
	C	14%
Student's college level	Freshman	5%
	Sophomore	0%
	Junior	23%
	Senior	73%
Gender	Male	71%
	Female	29%
Student's Age	20	5%
	21	36%
	22	45%
	24	9%

Average number of hours spent on homework and lab each week	Less than 1 hour	5%
	1 to 3 hours	30%
	4 to 6 hours	55%
	6 hours or more	10%

Table 1: Demographics

A fascinating aspect of the study was the researchers' observations and how these observations aligned with the student's opinions from the questionnaire responses. The researchers observed the students during the experiment and reviewed the video, then met to discuss their observations. Finally, the researcher who taught the course served as the expert opinion, graded all programs using a detailed rubric and made observations about the quality of the results which was then discussed with the second researcher.

The method used to determine program result quality in this research study (expert opinion) was similar to 19% of the studies identified by Salleh, et al. (2011) in their review of pair programming studies. The two commonly used methods were 1) expert opinion and 2) academic performance measures (Salleh, et al., 2011). Although, some performance measures were collected the resulting grades were not compared, instead the comments and general results of success and errors was summarized in these results. Two programs could not be graded, one from a pair and one individual since they did not turn in their projects using the suggested method so the results were lost. Another pair turned in the project but it was in the wrong format and could not be opened, this resulted in a total of 16 participants, four individual programmers and six pair programmers.

Of all the program results, only one ran flawlessly without error and produced correct results, this was produced by a pair. All other pairs and individuals created the *form* correctly, however, there were problems in the coding. Two pairs scored high on the grading rubric with one program running successfully as noted above. The other program was coded very well, ran and produced some results but eventually crashed due to an incorrect numeric format. The remaining pairs scored low, with programs either crashing or freezing. The exception was a program by one pair that ran and produced results but was not created according to the requirements and instead was a very simplistic program with no sub-procedures. None of the programs completed by individuals worked, they either crashed or froze.

Overall, the programs created by the pairs were structured well and included advanced concepts such as input validation and use of sub-procedures, some of which were started but not completed in most cases. A couple of individual programs created an area for the sub-procedure but did not include any code. The problems with sub-procedures possibly meant the students understood the concepts that were needed but could not complete the implementation of it due to lack of knowledge or time constraints. It appears the pairs did a little better than the individuals with the high level concepts but they were still mostly unsuccessful. This may have occurred because of the time constraint of 75 minutes for the lab experiment. However, it is encouraging that the students knew what was needed although they did not complete the execution of the concept. It was discouraging to see that some of the individuals and some pairs had trouble with two low-level concepts, i.e., using the 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 and was repeatedly used in 6 individual programs.

The researchers observed some interesting behaviors. First, getting the students to work as pairs instead of two individuals was a little challenging as seen in the first 20 minute session where some drivers were observed driving and navigating at the same time. Additionally, a couple of students were not focusing, looking around at what other students were doing. In general, the individuals appeared to be slightly behind in progress compared to the pairs. One individual had problems with their computer and had to begin again. Another individual was struggling and asked to borrow a book even though the experiment was almost over. Another struggling individual appeared to disengage; making conversation with a nearby individual on un-related topics.

Some pairs had problems. One pair started off badly, and they were never able to recover. Where most pairs decided who would begin as the driver, a member of this pair took over the keyboard and began driving. The navigator in this pair was unhappy about that action and made a comment to the researcher. Consequently, this pair did not talk to each other throughout the entire experiment and in fact when each took a turn they worked alone. This pair never worked as a cohesive unit and did not turn in their result in the end. Additionally, one of the partners disengaged when he was not driving, i.e. looking around, using his cellphone or appearing to take a nap while waiting for his turn again. A different pair had a partner who dominated the process and was somewhat condescending to their partner. However, this pair continued to make progress.

All other pairs appeared to be working well. During the second 20 minute session the learning curve for the process appeared to have been conquered and things began to improve. Several pairs were observed being very communicative and progressing well. One pair seemed to really enjoy the process and were working together, communicating well 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 before the end of the session. 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, be engaged, actively work together and exchange ideas. This is especially refreshing since the instructor had previously observed some of these same students stressed while trying to program individually during the course.

Although researcher observations are important, student perceptions have significant bearing on this research. Table 2 below shows results of some of the survey questions related to the students experience with the experiment. Overall, the results are in favor of pair programming because of perceived better results and a better experience which is consistent with results found in studies of CS/SE students (Salleh, et al., 2011). The first five questions show the majority of the participants felt a partner made the assignment easier. The majority of participants felt the pair programming process was beneficial, i.e. making them more confident about their code, enhancing problem-solving and improving unclear concepts by working with a partner. The next set of questions dealt with the pairs' opinions about how evenly matched they were with their partner. Results indicated 25% felt they were not evenly matched but 75% felt they were. This is important since the pairs were randomly selected. Despite this, only 15% felt pair programing was not as successful as individual programming while 85% felt pair programming led to more success. Finally, 75% of students felt pair programming should be a part of programming classes and 85% indicated they would recommend pair programming to other students. Most of the students felt the time limit was a constraint just as the researchers did, with 60% believing there was not enough time and with 40% feeling the time was sufficient.

Question	Strongly Disagree	Disagree	Agree	Strongly Agree
Having a partner made it easier to complete assignments.	5%	30%	35%	30%
Having a partner made me feel more confident that the code was reliable.	5%	15%	60%	20%
It was helpful to discuss programming problems and solutions with a partner.	5%	10%	55%	30%
Having a partner is beneficial for learning to read another programmer's code.	5%	5%	65%	25%
It was easy for me to get my pair programming partner to answer my questions.	5%	5%	65%	25%
My partner and I were equally matched in terms of the pace at which we solved programming assignments.	5%	20%	60%	15%
I am comfortable being the "driver"	5%	5%	48%	43%

I am comfortable being the "navigator"	5%	10%	50%	35%
Pair programming leads to more success than individual programming.	5%	10%	50%	35%
Pair programming should be part of every class that requires programming assignments.	5%	20%	45%	30%
I would recommend pair programming to other students.	5%	10%	45%	40%
There was enough lab time for the programming exercise	25%	35%	30%	10%

Table 2: Sample of Survey Question Results

Conclusion

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. The researchers believed more time was needed after grading the resulting programs and more than half of the students indicated in the survey that the time limit was too short. Another limitation common in surveys was the self-reporting of students on their own behavior. Also, the small sample size, although common in pair programming experiments may have made it difficult to generalize the results.

The researchers concluded the benefits were very similar to those found with CS/SE studies. Students appeared to enjoy the process of creating the program much more than those who programmed individually. The researchers also concluded from their observations and the student survey responses that students in pairs shared ideas which helped them to learn from each other. Sharing of ideas and learning from your 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 which we believe may be possible using pair programming techniques. This is supported by the results showing pair programming may be beneficial to the students' ability to create programs using high-level concepts. We hope this type of pair programming experience would also prepare the students for working collaboratively in the workplace where pair programming is being used as a part of extreme programming methods (Muller, 2006). Although a few students did not like working in pairs, this percentage was very small.

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. The researchers have plans to continue this research theme in the future by conducting a different pair programming experiment. Then the researchers will make a determination about altering the course to include pair programming as a part of the course methods.

References

Cockburn, A., & Williams, L. 2001. "The Cost and Benefits of Pair Programming," in *Extreme Programming Examined,* G. Succi, & M. Marchesi (eds.), Boston: Addison-Wesley Longman Publishing Co., Inc., pp.223-243.

Hoisington C. 2014. "Microsoft Visual Basic 2012 for Windows, Web, Office, and Database Applications: Comprehensive**",** Stamford: Course Technology, Cengage Learning, pp.1- 901.

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), pp. 90-95.

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), pp. 335-344.

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

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), pp. 509-525.

Slavin, R. E. 1999. "Comprehensive Approaches to Cooperative Learning", *Theory into Practice,* (38:2), pp. 74-79.

Williams, L. A., & Kessler, R. R. 2000. "The Effects of 'Pair-Pressure' and 'Pair-Learning' on Software Engineering Education," in *Proceedings of the 13th Conference on Software Engineering Education & Training*, Austin, TX, pp. 59-65.

Wray, S. 2010. "How Pair Programming Really Works", *IEEE Software*, (January/February), pp. 50-55.