https://doi.org/10.48009/1_iis_2013_169-179

Issues in Information Systems
Volume 14, Issue 1, pp.169-179, 2013

THE EFFECT OF SOCIAL ACTIVITIES ON PAIR PROGRAMMING: RESULTS FROM AN EXPERIMENT

Hezekiah J. Phelps, Washburn University, hezekiah.phelps@washburn.edu
Jacob Dobler, Washburn University, jacob.dobler@washburn.edu
Zachery T. Glenn, Washburn University, zachery.glenn@washburn.edu
Dr. Wenying Sun, Washburn University, nan.sun@washburn.edu

ABSTRACT

Pair-programming is a programming method where two people use one computer to work simultaneously on the same programming assignment. Previous research suggests pair-jelling plays an important role in how pairs perform. Pair-jelling is where the pair engages in activities to solidify and enhance their partnership. However, little empirical research has been done in the area of pair-jelling. In this paper, we explain the theory behind working in pairs, and why it is important for a pair to have effective communication. We also conducted an experiment to investigate the benefits of pair-jelling by comparing the quality of a programming assignment between two groups of pair programmers, one of which was engaged in a pair jelling activity. Our results suggest that there was no major difference in the time to complete the project and the quality of the program between the two test groups. This could be due to a multitude of reasons, including program complexity, social activity type, and time constraints.

Keywords: Pair Programming, Pair-Jelling, Social Interaction

INTRODUCTION

Programmers use a variety of techniques and strategies when working on software projects. One commonly adopted strategy, especially in agile development, is pair programming. Pair programming is one of the ten principles in Extreme Programming which is a popular version of agile software development. Pair programming is when two programmers work side-by-side at one computer collaborating on the same design, algorithm, code or test [10]. The programmer at the keyboard is called the "driver" and the programmer "watching" from the side is known as the "navigator". The driver is in charge of typing the code, compiling the code, and switching between applications during the development process. The navigator is responsible for evaluating the driver's code as it is being written, checking for any syntax errors or formatting faults, as well as offering alternate approaches to the design of a program. It is important to have the two programmers take turns typing, so neither one will become too complacent in their role. One requirement for a successful pair programming venture is a constant flow of communication in the form of ideas and suggestions between the two programmers. It appears that many businesses have used this approach on large projects and claimed that it has helped, but there is a lack of quantitative results. Some experiments have been done to show that pair programming is better with time management and with overall quality [6, 9]. The result of economic studies show that pair programming makes sense *economically* when the market pressure is high [8]. For large scale software projects, there are few drawbacks in exploring the approach of pair programming.

Judging from previous studies, it appears that having two programmers sit at one computer and collaborate with each other on one project improves the time, efficiency, and quality of the project. The simplest way of devising why this is would be to look no further than the old adage "Two heads are better than one." It would seem fairly obvious that the combined knowledge of two programmers can help deliver a higher quality program at a more rapid speed, but it would also seem that there may be some initial drawbacks to this approach. One of the drawbacks would be that the programmers may not know each other very well at all, and therefore may not feel too comfortable working together. One way of improving the relationship between the two programmers is an activity called pair-jelling. Pair-jelling is the process in which the pair of programmers engages in activities so they can get comfortable with each other. It is the time needed for a conditioned solo programmer to become a pair programmer [10]. It is claimed that pair-jelling is an effective tool for long term programming or other group work. Once a pair jells, future jelling is considered irrelevant and thus increasing the efficiency of work done by pair programmers. Over time, programmers learn the strengths and weaknesses of each member, and with this knowledge they can adjust their activities to exploit strengths and avoid weaknesses [10].

However, there is a lack of empirical evidence to show that pairs which are jelled from the beginning will perform better than groups who go into the project not knowing each other well at all. Therefore, the purpose of this research is to answer the following research questions:

1) Will a pair who engages in a social activity before working on a project communicate more effectively during the project than a pair who does not engage in a social activity?
2) Will a pair who engages in a social activity produce fewer errors and have a higher completion rate in their programming assignment than a pair who does not engage in a social activity?
3) Will a pair who engages in a social activity complete their programming assignment faster than a pair who does not engage in a social activity?

We answered these questions based on the results from an experiment we conducted. The results of this study have the potential to influence further research into the area of pair programming.

The rest of the paper is organized as follows: The Literature Review section summarizes and briefly discusses some of the previous research that is relevant in the area of pair programming. The Group Theories section covers the theory behind why groups function the way they do, and important elements that influence the way a group behaves. The Hypotheses section lists the hypotheses we came up with, as well as the supporting arguments for these hypotheses. The Methodology section summarizes how we conducted our experiment, including how we collected and analyzed important data from this experiment. The Results section presents our results and explains what statistical relevance they have, if any. The Discussion section extrapolates our thoughts on these results, why we think they came out the way they did, and what we may have done differently in retrospect. Finally, our Conclusion section presents our closing thoughts on our experiment and what we learned from this project.

LITERATURE REVIEW

Several experiments have been found regarding the process of pair programming. The experiments varied between the experimental group (whether they used students versus professionals), and on duration (whether it was a couple assignments during one semester versus several assignments and groups over a year or more). Overall, the consensus on pair-programming is that it is effective and powerful, but many programmers who have years of experience working alone are initially hesitant to work in pairs. For example, an experiment was conducted to compare individuals working on a programming problem and teams of two working on the same problem. The finding was that the programmers in pairs took an extra 60 percent minutes more to complete the entire assignment, however to complete the actual task it took the pairs 40 percent less time with more efficient algorithms and code [10].

Although most experiments show that pair programming can be beneficial, it may not always be the case that the programmers will be working together all the time. This may be due to various reasons including health issues and timing conflicts. Since this is the reality of most work environments, programmers have "prioritized which parts of the development cycle are most important to work together, which can be done separately, and what to do with the independently developed work when reuniting" [10]. As explained by Williams, two steps in which it is critical for pairs to work are the pair-analysis and pair-design stages. These are the stages in program development when programmers build an understanding of the design process, analyze the requirements, and create a plan on implementation processes. The advantage of having pairs work together during these stages allows for more thought out designs and less margin for error in the finished product. The two programmers can bounce ideas off each other and have more of a chance of catching any possible errors. The next stage of pair programming is pair-implementation. This stage is slightly less critical than the previous two, as programmers can choose to implement on their own at any point. Depending on the business requirements for the program and the simplicity of the implementation, it is sometimes easier to work individually at this stage. However, working together for this stage in the driver/navigator format is an advantage for larger projects. This way the navigator can critique, catch errors, and think of alternative methods for completing the project while the driver types up the actual code. The pair can switch roles so they can keep as many options flowing between the two as possible. If the pair decides to do the programming solo, they can meet up after the pair-implementation process to come up with test cases for their program, and this leads into the final sage, pair-testing. This is the least critical stage, as long as the programmers meet to create these test cases. The actual testing can be done individually, and in the event of any errors being found, the programmers can meet to resolve the problem [10].

Another study shows the effectiveness of pair programming based on the complexity of the software system and the knowledge and expertise of the programmers. The experiment used 295 Java programmers from companies in Norway, Sweden, and the U.K. [2]. Some of the measurements held for this experiment were the duration necessary to complete the assignment, the programming hours needed between the pairs compared to an individual, and the correctness of the assignment by each group. The programmers were given a training task to understand the experiment process, a pretest task to gauge their level of expertise, and four main tasks to measure the effects of the pair programming against individual programmers. Overall, the duration of the pairs was 8 percent less than that of the individuals to complete the assignment correctly. This duration corresponds to an 84 percent increase in effort by pairs and a 7 percent increase in correctness compared to individuals [2]. The experiment shows slightly different results when looking at the difficulty of the task. When given a simple task, the pair programmers had a 20 percent decrease in duration when compared to individuals, but a 60 percent increase in effort. Surprisingly, it showed a 16 percent decrease in correctness compared to individuals. When given a more difficult task, pairs took a 6 percent increase in duration, a 112 percent increase in effort, and a 48 increase in correctness [2]. This experiment shows finer details about pair programming in that with a simpler programming task, duration is decreased. However with a more difficult programming task, correctness is increase greatly [2].

From the previous examples we can see the implied effectiveness of pair programming, but we also want to look at techniques that will help a pair communicate better from the beginning of a project, and this is where we turn our attention to pair jelling. Indeed, there is an initial adjustment period in the transition from solitary to collaborative programming. In industry, this adjustment period has historically taken hours or days, depending upon the individuals [10]. An experiment conducted in the area of pair jelling studied the duration it took for a pair of programmers to complete an assignment compared to an individual - however it then studied the durations for a second assignment with the same pairs. For the first assignment, the pairs finished in shorter elapsed time and had better quality, but they took, on average, 60% more programmer hours to complete the assignment when compared to the individuals. After the adjustment time, this 60% decreased dramatically to a minimum of 15%. The end of the second assignment marked an important milestone -- all students reported that they had overcome their constant urge to grab the mouse or keyboard from their partner's hands! [10]. We can see from this study conducted by Jeffries that it takes a certain amount of time, and most likely a certain number of programming projects, before a pair will feel comfortable enough that they will overcome the initial loss of programming time to adjust.

GROUP THEORIES/THEORETICAL FOUNDATION

Working in groups can be beneficial in the long run. A key factor in group dynamics is the outcome expectation for group members. Group members extend task persistence when they have a positive expectation on the outcome of their group work [7]. There may also be extra motivation found in pair programming, as a person may strive to work harder towards their goals as not to let their teammate down. The kind of mutual trust and respect that may build among teammates has the potential to translate to a higher quality product than if a person worked on it alone.

Another requirement of successful group performance is a person building a social identity with their teammate. According to Henri Tajfel, group members share a common identity with one another. They know who is in their group, who is not, and what qualities are typical of insiders and outsiders. The perception of themselves as members of the same group or social category - this social identity - creates a sense of *we* and *us*, as well as a sense of *they* [1]. There is also the idea of interdependence - members must depend on one another; their outcomes, actions, thoughts, feelings, and experiences are determined in part by others in the group. "Some groups create only the potential for interdependence among members. The outcomes of people standing in the queue at a checkout counter in a store, audience members in a darkened theater, or the congregation of a large church are hardly intertwined at all. Other groups such as gangs, families, sports teams, and military squads create far higher levels of interdependency since members reliably and substantially influence one another's outcomes over a long period of time and in a variety of situations" [5].

The previously mentioned group dynamics are good to have, but they may not get a team very far if they don't have the appropriate work environment to flourish, or enough control over the project to make important and meaningful decisions together. Experiments have demonstrated clearly that the productivity of work groups can be greatly increased by methods of work organization and supervision which give more responsibility to work groups, which allow for fuller participation in important decisions, and which make stable groups the firm basis for support of the individual's social needs [3, 4]. A supervisor always needs to have a certain amount of control over what a team is working on so that they may make sure that the development process is moving in the right direction (rather than spinning in circles). However, letting groups make certain decisions during the development process may pay off in the long run, because it gives them a catalyst to build upon the aforementioned trust and respect.

So how do we tie all of this together in our experiment? We believe that the social activity will help to build a stronger social identity among members than the groups who do not participate. These groups should feel more comfortable with each other heading into the project. Our social activity requires interdependence. However, it is possible for one person to do all of the work on the programming project, while the other person is silent. Therefore in order to enforce interdependence we will have the programmers switch "roles" every ten minutes. We also choose to give the pairs the initial specifications for the programming project, but allow them to make all appropriate decisions on how to proceed with the development process with their teammate. We hope that using these guidelines for interaction will help us obtain some meaningful communication among participants.

HYPOTHESIS

We have used the previous research to develop three main hypotheses we wish to test through our experiment. We used the data we collected from a pair's finished (or unfinished) program to test these hypotheses.

Previous studies about pair jelling suggest that the more jelled a pair is, the more comfortable they are together, and the better they will communicate. This means more natural dialogue and better conversation flow. We believe that introducing the pair to each-other in a setting where they can openly converse *before* jumping right into the programming assignment will affect their conversation throughout the duration of the assignment. This can also be affected by the teammates building a social identity with one another. Therefore we hypothesize:

H_1: *Pairs who engage in social activities will converse more (have a higher conversation rate) than pairs who do not engage in social activities.*

We form the following hypothesis on the basis of the first; we believe that groups that converse on a more regular basis will be able to discuss and solve problems more effectively during the development process. This will in turn affect the overall quality of the program, whether it is more accurate output, or cleaner and more efficient code in general. Though the programming experience of both individuals in the group will also factor into this hypothesis, we believe that if the experience is evened out among the participants (which it should be, considering they are all in the same class) then stronger communication will be the deciding factor. Therefore we hypothesize:

H_2: *Pairs who engage in social activities will produce fewer defects in their program than pairs who do not.*

The following hypothesis is also based upon communications. However it is also based on the theory that a group that has already formed some sort of social identity through the activity will "hit the group running", or take less time during the project to learn about their partner than they would otherwise. They may apply some of the things they learned about their partner during the social activity during the programming assignment. Since we are comparing pairs to pairs, and not pairs to individuals, we can only assume that the extra time the pair gets to spend together before attempting the program can only help them complete it quicker. Therefore we hypothesize:

H_3: *Pairs who engage in social activities will complete their program faster than pairs who do not.*

METHODOLOGY

Our experiment's sample contained solicited classmates, coworkers, and friends who had at least a basic understanding of java programming. Our sample size was 46. A sample size of 46 participants allowed us to have 23 pairs. 12 of these pairs participated in both the social activity and the programming assignment, while the other 11 pairs only participated in the programming assignment. These 46 participants were split up into four different experimental sessions, which made it much more manageable for us to conduct and regulate. Our allotted time to conduct the experiment was 1 hour 15 minutes, in which we used 15 minutes for the social activity and 1 hour for the programming assignment.

The participants knew ahead of time that they would be taking part in a programming experiment for which they would receive extra credit. However the specifications of the programming assignment, including the fact that they would be working in pairs, and some might be involved in social activities, were not revealed to them until they showed up to class.

Upon arrival, participants were divided into two groups randomly. Each participant drew a colored number out of a container. If a participant drew a blue number three, he/she was paired up with the other blue three. The color of the number decided which pairs would participate in the social activity. Blue numbered pairs participated in the social activity. Participants who drew green numbers found their partner and began working on the programming assignment. Assigning numbers this way proved to be an effective method all while avoiding bias or favoritism in the eyes of our participants. Each workstation in the computer lab was assigned with a corresponding color/number combination to keep track of pairs and workstations. Workstations were labeled so that no pair would be sitting by a pair of the same color.

Once divided, the pairs participating in the social activity were escorted to a nearby remote location while the other pairs began the programming assignment. Our social activity was limited to fifteen minutes, in which each pair spent their first five minutes interviewing each other. Each participant was given a piece of paper with predefined interviewing questions on it. Participants used the questions as a question bank, but were free to ask other questions that they were comfortable with discussing between each other. We chose to have these pairs conduct introduction interviews with predefined questions because this would serve as an icebreaker, allowing each partner to become comfortable with the other. Predefined questions were chosen to keep the interview short and focused. Once the introduction interviews were complete or five minutes had elapsed, we started the social activity.

The basis of our social activity was that it needed to engage both members of a team so that they strived to complete a common goal by interacting with each other. This could have been either a physical activity such as a game of 2-on-2 basketball, or a mental activity such as solving a puzzle. How the team interacted with other teams is far less important than how the two members of the team interact with each other. Guesstures was chosen as the social activity because it would force our pairs to continue to interact verbally but also through facial expression and body language. The game will also push the pairs to work together towards a goal - the goal of winning. During a round of Guesstures, one member selects four “action” cards and places them on a timer. When the timer begins, the “actor” begins to act out the words on the “action” cards while their teammate tries to guess. The physical communication from the “actor” and the verbal communication from their member become more frantic as time runs out. After completing the social activity, the pairs were brought back to the initial location and began working on the programming assignment.

The programming assignment we decided on was influenced by several different factors. First of all, it needed to be challenging enough to require the pair to collaborate; however it couldn’t be too advanced as most of our participants were likely to have minimal programming experience. It also needed to be something that could be completed over the course of one hour. Full completion was not required, though we used completion as a unit of measure of how effectively the pair worked together, amongst other things. We also needed to figure out how much freedom to give our pairs while working on the assignment. We could have either given them a very specific list of programming steps, or we could specify what the desired results were, and let each pair start at point A and arrive at point B by their method of choice. All pairs had access to the internet, as this was a good source of information in completing the project when neither group member knew what to do. How the pair interacted while looking up information on the internet was as much value to us as the programming work itself.

Each pair was given a sheet of paper detailing the programming assignment. At the top of each assignment sheet was a description of the driver and navigator roles of pair programming, and instructions requiring the pair to switch roles every ten minutes. At each ten minute interval, we announced and enforced our requirement of each pair to switch roles.

Throughout the duration of the experiment, we collected data individually on how the pairs communicated with each other. This data was part observation and part completion. Observation was used to link our findings together and identify reasons for why some groups did better than others. The criteria with which we graded each group are listed as four measures of conversation duration, flow, quality, and topicality. Conversation duration measured how long a pair held their conversation. Conversation flow measured how active each partner is in their pair’s conversation. Conversation quality measured whether the pair’s conversation consisted of questions and responses, suggestions, and feedback. The conversation topicality measure was determined by comparing a pair's conversation topic to the topic of our programming assignment. Each measure was graded with a score ranging from 1 (low), 3 (medium), and 5 (high).

The more important form of data we collected was how complete each group's program was and how accurately it produced the desired result. In order to judge this we needed to complete the program ourselves. During completion, we tried to find the most complete and efficient way to produce the desired result. Since this assignment could have been completed several different ways, there was no "right" or "wrong" answer. Although, some answers were more efficient than others. Professionalism in the program, including comments in the code and documentation of the program were looked for and graded on.

The five measures we used to grade each program were assignment completeness, documentation, readability, syntax errors, and output errors. When grading assignment completeness, in the program, we looked for a completely working program that produced the desired output. Documentation measured the amount of explanation within the code. Readability measured how well the pair organized their program code and if it was easy to follow. Syntax errors measured the number of errors within the code that prevented the program from compiling and running. Output errors measured the quality of the output, whether the output contains the desired results, and looked neat. The scores ranged from 0 (requirements were not met), 1 (some requirements were met), and 2 (all requirements were met).

After the experiment was finished, we individually graded each programming assignment. Once all the grading was complete, we averaged our observational scores as well as the scores from each of the programming assignments.

RESULTS

We used 46 students who, at the time of the experiment, were in beginner Java programming courses (Intro to Structured Programming and Contemporary Programming Methods). The students were split into 12 social groups and 11 non-social groups. Many of the students have had experience in programming development outside of the courses they were taking. Below is a table showing the demographics of the students who participated in the experiment.

Table 1: Participants' Demographics

Gender	**Age**	**Prior Programming Experience**	**Programming Classes**
Male: 38 82.6%	Min: 18	Yes: 27 58.7%	1: 18 39.1%
Female: 8 17.4%	Max: 66	No: 19 41.3%	2: 6 13.0%
	Mean: 25.63		3: 17 37.1%
			4: 3 6.5%
			5+: 2 4.3%

Below is a table that shows all of our final scores for the social group and non-social group. We used an independent sample t-test to calculate our mean (average scores), standard deviation (the variation from the mean in our scores) and statistical p-value (significant difference). The p-value determines whether the averages between the two groups are significant, and represents that with a number value. A result of 0.05 or smaller shows that the difference is truly significant. When taking this measurement, we assumed that there was an equal variance between the two groups since all the participants were in the same class and on the same level of knowledge about the topic. In the table, group 1 shows the results for our social group and group 2 is our non-social group.

Table 2: Results on Independent Sample T-Test

Hypothesis	Measure	Group	Mean	Standard Deviation	P-Value
H_1	Conversation Duration	1 2	3.417 3.182	1.240 1.167	0.646
	Conversation Flow	1 2	3.500 3.545	1.087 1.213	0.925
	Conversation Quality	1 2	3.833 3.545	1.193 1.128	0.559
	Conversation Topicality	1 2	4.833 4.636	0.577 0.674	0.459
H_2	Assignment Completeness	1 2	1.417 1.364	0.668 0.674	0.852
	Documentation	1 2	0.250 0.273	0.452 0.646	0.923
	Readability	1 2	1.833 1.818	0.389 0.404	0.928
	Syntax Errors	1 2	1.917 1.818	0.288 0.603	0.618
	Output Errors	1 2	1.250 1.182	0.621 0.750	0.814
H_3	Duration (minutes)	1 2	55:750 56:455	7:136 5:768	0.798

In regards to our first hypothesis, the following are the results between the two groups that reflect our findings in their communication. For the conversation duration, we saw that the social group averaged a 3.417 score on our scale and the non-social group averaged 3.182. Conversation flow resulted in the social group scoring a 3.500 average and the non-social group scored a 3.545. The conversation quality showed that the social group averaged a 3.833 score and the non-social a 3.545. Finally, the conversation topicality resulted in the social group showing a 4.833 score and the non-social group a 4.636. Scores between both groups are very close together. There are differences between the two groups, but the difference is not statistically significant. Therefore, H1 is not supported.

In regards to our second hypothesis, the following are results that reflect each group's work on the programming assignment. For actual completion of the assignment, we had six social pairs and five non-social pairs finish the program completely. On average, the social group was scored a 1.417 for completion and the non-social group a 1.364. For documentation, the social group averaged a score of .250 and the non-social group averaged .273. Readability results show that the social group averaged a score of 1.833 and the non-social group averaged 1.818.

For syntax errors, the social group scored a 1.917 and the non-social group scored 1.818. Finally, the output errors show that the social group had a score of 1.250 and the non-social group had a score of 1.182. H2 is not supported

In regards to our third hypothesis, we timed each pair for one hour to complete the assignment. Five of our social pairs and four of the non-social pairs finished completely before the hour had passed. The average time for the social group to finish the assignment was 55:750 minutes, and the non-social group's average time was 56:455 minutes. H3 is not supported.

DISCUSSION

Our first hypothesis is pairs who engage in social activities will converse more (or have a higher conversation rate) than pairs who do not engage in the social activities. The scores for the communication categories do not show any significant difference between the two groups. Because the scores are so close between the two groups, our first hypothesis was not supported with this experiment. As time progressed during the experiment, we observed that conversation amongst all the pairs started off steadily, but then increased greatly as the experiment progressed. The majority of the pairs stayed on topic during the experiment. The conversation duration and quality were consistent within the groups. All the pairs sustained good communication throughout the entire experiment and we noticed that the non-social group had a couple more pairs with better flow of conversation. This may be because the non-social groups did not participate in the social activity and needed to compensate the lack of jelling with more conversation to complete the assignment. Another reason for why we believe the communication scores are so close together is because our participants were all from the same college class. We conducted our experiment with only a month left of the school semester, and by then students in the class had ample time to get to know one another. The students may already have had an understanding of one another's abilities and knowledge with programming because all the students were somewhat familiar with one another prior to our experiment. Our social activity did not have as much influence on their teamwork because the participants were already comfortable (or already jelled?) with each other.

Our second hypothesis is pairs who engage in social activities will produce fewer defects in their program than pairs who do not. The categories with which we graded the program assignment show that there is not much significant difference between the two groups. Because the scores for our programming assignment are close between the two groups, this means that our second hypothesis is not supported. All of the pairs had little to no documentation in their program except for one pair in our non-social group. We noticed that due to the time limit on the program assignment, most groups decided to hold off on documentation in the code and only add some if time permitted at the end of the assignment. Almost all of our groups finished the first half of the program assignment and several of those pairs got close or finished the second half of the program assignment. Most of the pairs produced programs that were easy to read and only a couple programs had syntax errors that prevented the program from compiling and running. The output of the programs differed from pair to pair. Many of the programs gave an output that met our requirements. A couple of the outputs gave produced some of our requirements and lacked in neatness or clarity. A large reason why we believe our scores for the programming assignment categories are so similar is again because of our selection of participants. Since we used students who are in lower level programming classes, most of their experience with programming is within the classes. This means that they each might have been taught to program in a similar fashion.

Our third hypothesis is pairs who engage in social activities will complete their program faster than pairs who do not. Because the groups took about the same amount of time to complete the assignment, our third hypothesis is also not supported. We noticed that putting a time limit on how long pairs can work on the program assignment may have prevented the pairs from working together long enough to see any significant difference. Previous experiments show that groups who have jelled work better on more complex program assignment and perhaps our choice of program assignment did not provide enough of a challenge for the social pairs to fully utilize their jelling process.

Although all our hypotheses were not supported by our experiment, we can see from our previous research that conducting a social activity with groups could influence the performance of a pair. We noted possible reasons from our experiment as to why we were unable to show the same results. We speculate different approaches in preparing and conducting lab experiments in slightly different settings could provide diverse results.

CONCLUSIONS

Our experiment has given us much insight into how businesses have evolved with their program development processes by pairing two programmers together to combine their skills to complete a programming assignment. Pair programming has allowed many advances in software development and continues to pick up popularity through businesses. We learned through prior research that pairs of programmers who engage in pair jelling activities will show improved results in the quality, correctness, and time necessary to finish a program assignment. We also learned that in any group activity, if the pairs of the group have positive thoughts about how the group work will go, then the success in the group will be that much greater. Using these theories, we developed an experiment process that we thought would support our three hypotheses.

Through our experiment, it was our intent to show that a pair of programmers who engaged in a social activity prior to working on a programming assignment would have improved communication and programming quality compared to a pair of programmers who are not engaged. We conducted an experiment and tested whether groups with pair-jelling would work better than groups without the pair-jelling process. After analyzing the data, we found there was no significant difference between the two groups.

However, we do not think that our results diminish the importance of social activities between the pairs. Our results are true given the specific subjects, constraints, and programming environment. With changes to our experiment process, we could show that the pair-jelling process is a necessary aspect to pair programming. We suggest one or more of the following future research to be done. An increase in the sample size or a more diverse sample size could show more variance between the groups. Another variable change that could be made is allowing more time for the completion of the programming assignment. Increasing program complexity may also present a different result. Another variable would be to increase the duration and frequency of engaging pairs in the social activity. If the number of programming assignments is also increased along with the frequency of social activities, we could see the social group improve with each programming assignment. Finally, since we used students for our experiment, we could have conducted our experiment towards the beginning of the semester before they had an opportunity to get to know each other through the progression of the class. If one or more of these changes are made, we believe our experiment would show the power of pair-jelling and the effects of social activities on pair programming.

REFERENCES

1. Abrams, D. (2005). The Social Psychology of Inclusion and Exclusion. Psychology Press, Feb 1, 2004.
2. Arisholm, Erik, et al. "Evaluating pair programming with respect to system complexity and programmer expertise." Software Engineering, IEEE Transactions on 33.2 (2007): 65-86.
3. Cartwright, D. (1951). Achieving Change In People: Some Applications of Group Dynamics Theory. Human Relations. 4(1), Sage Publications.
4. Coch, L., & French, J.R.P, Jr., (1948). Overcoming Resistance to Change, Human Relations, 1(4), 512-532
5. Forsyth, D. (2009). Group Dynamics. Wadsworth, Cengage Learning 2009.
6. J. Nosek. The case for collaborative programming. *Communications of the ACM,* 41(3):105-108, Mar. 1998.

7. Jung, D. I., & Sosik, J. J. (1999). Effects of group characteristics on work group performance: A longitudinal investigation. *Group Dynamics: Theory, Research, and Practice, 3*(4), 279-290. doi: http://dx.doi.org/10.1037/1089-2699.3.4.279.
8. Müller, M.M., & Padberg, F. (2004). An Empirical Study about the Feelgood Factor in Pair Programming. Software Metrics, 2004. Proceedings. 10th International Symposium on.
9. Nawrocki, J., & Wojciechowski, A. (2001). Experimental evaluation of pair programming. *European Software Control and Metrics (Escom)*, 99-101.
10. Williams, L., Kessler, R. R., Cunningham, W., & Jeffries, R. (2000). Strengthening the case for pair programming. *Software, IEEE*, *17*(4), 19-25.