Examining the Impact of Pair Programming on Female Students

Chih-wei Ho[1], Kelli Slaten[2], Laurie Williams[3], and Sarah Berenson [4]

ABSTRACT

There has been low representation of women in Computer Science. Numerous studies have been conducted to identify the cause of this under-representation and to provide suggestions to improve the situation. Still not much progress in attracting women to computer science has been observed. The research discussed in this paper was done during the pilot study phase of a three-year project about women in information technology field. During the first semester of this project, pair programming was used in a junior/senior Software Engineering class at North Carolina State University. The goal of this research is to examine the effect of pair programming on female students. We interviewed three female students and analyzed all female students' project retrospective reports. Theoretical models were developed to describe (a) the source of project enjoyment, (b) context that influenced female students' study habits, and (c) the effectiveness of pair programming. The cause and effect of each component of the theoretical models were identified and are illustrated with narrative data.

1. INTRODUCTION

Women's participation in Computer Science (CS) has always been low. Numerous studies have been conducted to identify the cause of this under-representation [e.g. 7 and 12], and the researchers have made several suggestions to attract more women to this field [e.g. 11 and 16]. Still, according to recent statistical data, female students are still a minority in CS education system [23, in 2002]. The low percentage of women in CS education programs leads to the low supply of women professional workforce. We need to take steps to encourage more women to join this field.

Pair programming, whereby two programmers work at one computer on the same programming task, shows several promising properties [33] for educational purpose. A qualitative study was conducted to examine the effect of pair programming on female undergraduate CS students. Pair programming was used in a junior/senior Software Engineering course (CSC326) at North Carolina State University (NCSU) in fall 2003. In the middle of the class three students were interviewed. At the end of the class, the students wrote a retrospective essay of their experiences. Qualitative research methods, including grounded theory, were used to analyze the female students' interviews and their retrospective essays on the final project. The purpose of this phase of the research is to develop a theory about the effect of pair programming on female students, guided by the following two questions:

1. How were the study habits of the female students affected by pair programming?

[1] Chih-wei Ho, North Carolina State University, Department of Computer Science, cho@unity.ncsu.edu
[2] Kelli Slaten, North Carolina State University, Department of Math and Science Education, kmslaten@ncsu.edu
[3] Laurie Williams, North Carolina State University, Department of Computer Science, williams@csc.ncsu.edu
[4] Sarah Berenson, North Carolina State University, Department of Math and Science Education, berenson@unity.ncsu.edu

2. How and why did the female students like (or dislike) pair programming?

The rest of this paper is organized as follows. Section 2 provides prior studies about gender issues and pair programming, as well as the research method utilized in this research. Section 3 gives a detailed description about the research approach, and Section 4 presents the research results. Finally, Section 5 concludes the paper and suggests future work.

2. BACKGROUND

In this section, I will present a literature review of studies about gender issues in the field of CS. Then, I will discuss the background of pair programming and its applicability in a classroom setting. Finally, information about the research method used in this study will be provided.

2.1. Women in Computer Science

There has been an ongoing concern about the low representation of women in the field of CS. Statistics shows that women are under-represented in academic CS, as shown in Figure 1. The graph shows the percentages of post-secondary degrees conferred to female CS students by degree-granting institutions from 1988 to 2001 [calculated from 23]. From the graph, we can see that the percentages of female CS bachelor and masters degree receivers have been consistently around 30% in the last decade, and the percentage of female CS doctoral degree receiver is even lower.

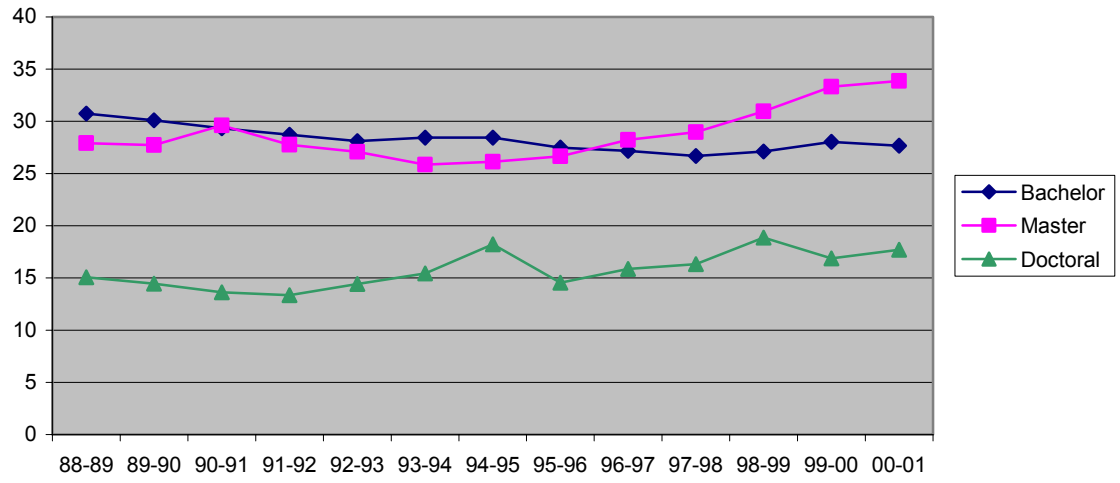

Figure 1: Female percentage of post secondary degrees in Computer and Information Science

For the purpose of comparison, Figure 2 shows the percentages of secondary degrees in all fields [calculated from 23, in 1990-2002] conferred to female students. The graph shows that there are increasing percentages of female degree receivers, an opposing trend to what is seen in CS. While approximately 50% of post-secondary degrees were awarded to women, female participation in CS has been low. Nevertheless, women's participation is important. Cohoon [12] emphasizes some reasons why we need participation of women in CS, including maintaining a supply for CS professionals, providing diverse viewpoints, and promoting gender equity. Actions must be taken to attract more women to CS field.

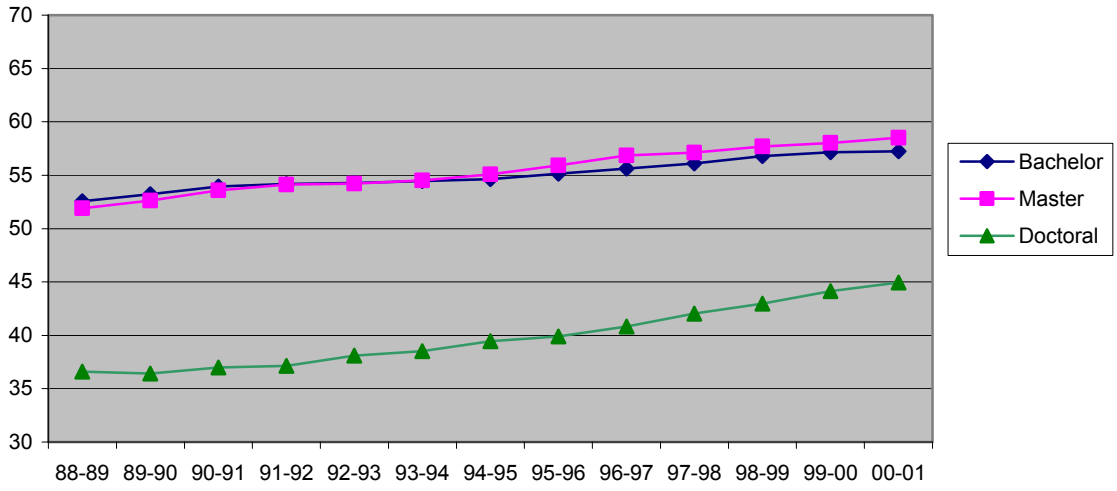

Figure 2: Female percentage of post secondary degrees in all fields.

Camp describes a pipeline shrinking problem concerning women in CS [9]. Camp notes that although 50% of high school CS classes were made up with women in 1993 – 1994, only 28.4% of the bachelor's degrees in CS were awarded to women. The number went even lower with graduate degrees: 25.4% at the master's level, and 15.4% at the doctoral level. Camp contends that computer scientists and educators have not done enough to make progress and that dramatic change needs to be taken. From the recent statistics data from National Center for Education Statistics, we can see that the pipeline is still shrinking today [23, in 2002]. A consequence of the low number doctoral degree in CS received by women is the low number of female faculty. According to the CRA Taulbee survey, fewer than 20% of newly hired faculties in year 2001-2002 were female [29]. The lack of female faculty results in the lack of CS role models for female students, which is reported as one of the reasons why girls do not pursue CS careers [16]. It is necessary to improve women's retention in CS education to balance representation of different genders.

Researchers have shown that men and women have different passions and interests in learning. Margolis et al. [20] point out that female students in CS are more interested in broader, people-oriented issues, and male students like CS because of their fascination with the machine. A survey of 567 first-year college students shows that female students are significantly more interpersonally-oriented, career-oriented, and family-oriented when compared with male students [6]. However, computing is widely stereotyped as a solitary, masculine activity. CS students are perceived to be very smart, spending their days working alone in front of the computer and talking about nothing but computer science [19]. The impression of this field is discouraging to female students.

Lack of confidence is also a reason that women stay away from CS. Beyer et al. indicate that women have less confidence in their computer capability [7]. Bayer conducted a survey of 56 students (24 females and 32 males) enrolled in a CS course at the University of Wisconsin-Parkside. The survey results indicate that although men and women thought male students outperform female students in CS, females CS students actually

have a higher GPA [5]. That men can do better in "masculine" majors is a misconception, but it also dispirits women from joining CS at the first place.

To address the problems, the American Association of University Women Educational Foundation Commission on Technology, Gender, and Teacher Education performed a survey of 900 teachers and qualitative research with more than 70 girls [1]. In the report, the commission makes several key recommendations for schools. The recommendations include the following:

- School software should not be specially designed for boys or girls.
- Provide positive role models.
- Change the negative stereotype of computing as a solitary activity.
- Engage girls in the "tinkering" activities of computing to stimulate deeper interest in technology.
- Design classroom activities for gender equity. Design group work and encourage multiple approaches for learning.

2.2. Pair Programming and Its Effectiveness in Education

Pair programming is a practice, whereby two programmers work side by side at the same computer, continuously collaborating on the same design, algorithm, code, or test [33]. One of the programmers is the *driver*, who has the control of the keyboard and the mouse, actively implements the program, and explains the implementation to his or her partner. The other is the *navigator*, who constantly watches the driver, reviewing driver's design, detecting driver's errors, and being a brainstorming partner. After a period of time (usually less than one hour), the programmers switch their roles. Pair programming has been practiced for a long time and has recently been popularized by Extreme Programming [3], an emerging software development method. Early evidence for the effectiveness of pair programming was only anecdotal. Recent researches find that pair programmers can produce code with higher quality without sacrifice of productivity [10].

Williams and Kessler point out seven behaviors that happen naturally when programming in pairs [33]. These behaviors make it possible for pair programmers to finish their tasks about twice as fast as solo programmers, with the use of approximately the same overall resource expenditure, and still generate higher quality products. These seven behaviors are:

1. *Pair pressure*: Williams and Kessler report that, in a junior-senior software development course, most students worked harder when paired because they did not want to let their partners down [31]. The students do not intentionally put pressure on their partners. Nevertheless, each feels the push to move forward.
2. *Pair negotiation*: The two programmers work together to solve a problem. They have different prior experiences but a common goal. The two have to negotiate to share a common approach.
3. *Pair courage*: The programmers give each other the courage to do something they might not do if working alone.
4. *Pair learning*: The programmers can learn from their partners' continual critique and review. Additionally, because the programmers work closely together, their

knowledge, including programming tips, design skills, tool usage, is transferred between them constantly.

5. *Pair trust*: Pair programmers work in a collaborative fashion. They learn to trust their partners to get the work done.
6. *Pair review*: When working in a pair, both programmers review their joint product continuously. This review technique has been shown effective and enjoyable.
7. *Pair debugging*: Debugging is a tedious and laborious task. However, if we can discuss the problem with someone, we might find new ideas and solutions.

Because of the promising properties of pair programming, some educators started using it in the classroom. Research conducted at the University of Wales indicates that students with low self-confidence seem to enjoy pair programming most, while students who think they have high skill level like to pair the least, especially when they have to work with students with less confidence [28]. Although the authors did not distinguish the students' attitude from their real performance in this research, it still shows that students with high confidence usually regard themselves as "lone rangers." However, another study done at NCSU indicates that the association between the self-esteem and the compatibility of pair programmers in a classroom is weak [18]. The difference may result from the inaccuracy of the students' self measurement of self-esteem.

Research conducted at University of California at Santa Cruz (UCSC) and NCSU have shown the effectiveness of pair programming in introductory CS courses [21, 22, 34]. Their studies show that paired students achieve higher performance and produce higher quality code. Furthermore, those studies also show that pair programming helps student persistence in computer science related majors. Sanders indicates that students who favor for pair programming think it enhances their learning experience, while those opposed to it think pair programming can pull back good programmers and should be practiced only with competent partners [25]. These studies about pair programming show that pair programming is a promising approach for computer science educators to use with their students. However, we need to explore the variety of students' responses to enhance the students' learning experience.

2.3. Qualitative Research

Qualitative research is widely used in the fields of education, nursing, sociology, and psychology, to explore the complexity of social or human problems. Denzin and Lincoln describe qualitative research as "multimethod in focus, involving an interpretive, naturalistic approach to its subject matter." To perform qualitative research, the researchers need to "study things in their natural settings, attempting to make sense of or interpret phenomena in terms of the meanings people bring to them." [14] The intent of our study is to examine the effects of different programming practices among the female students in an undergraduate level Software Engineering course. It is not possible to measure students' behaviors using quantified measurements without losing the accuracy of the meaning. Therefore, qualitative methods are used in this study. Two qualitative data collection methods are used:

1. Interview: In qualitative research, interviews usually involve unstructured and open-ended questions. The intent of interviews is to elicit views and opinions from the interviewees [13].

2. Document: Another source of qualitative data can be text in documents. Documents offer data that are more thoughtful than interview, because the participants can give more attention when writing a document. However, if the documents are not in a structured format, the researcher may need to make an effort to find useful information. [13]

Qualitative data can be analyzed via a technique called *coding*. Some qualitative data, especially more structural ones, can be quantified so statistical methods can be applied. Seaman recommends several coding methods for studies utilizing mixed (quantitative and qualitative) approaches [26]. In Seaman's definition, coding is the process of turning qualitative data into quantitative data. The quantifying process usually results in loss of information. However, the quantified data are more reliably accurate when they are restricted to "straightforward, objective information."

2.4. Grounded Theory

The majority of the data used in this research cannot be quantified. To preserve the complexity of the data, a qualitative research approach called *grounded theory* is used. Grounded theory was first described by Glaser and Strauss [15]. The spirit of grounded theory is that the theories can emerge from gathered data, in contrast to research in which data is gathered to support or refute a theory. The intent is to generate or discover a theory that is "grounded" in data from the field. In grounded theory, new theory comes from a small size of cases, exploring a wide variety of variables. This approach is suitable for this research because the impact of pair programming on female students is unclear.

Strauss and Corbin define a coding procedure for grounded theory [27]. Note that coding in this context is different from Seaman's definition. Here coding is the process that attaches labels to different categories (or themes). Strauss and Corbin define theory as a set of interrelated, well-developed categories. The interrelationships among the categories form a theoretical framework that explains the phenomenon existing in the collected data. The proposed coding procedure has three steps: open coding, axial coding, and selective coding. *Open coding* is the process of identifying the categories in the data and the properties of the different categories. *Axial coding* is used to connect the categories and to find their interrelationships. In the last step, *selective coding* identifies one or two central categories and forms a conceptual framework from which to generate a theory. Strauss and Corbin's coding process is utilized in analyzing the qualitative data in this research to examine the effect of pair programming on female students.

3. RESEARCH APPROACH

The focus of this research is to find out the female students' response to pair programming pedagogy. There are two questions guiding the research:

1. How were the study habits of the students affected by pair programming?
2. How and why did the students like (or dislike) pair programming?

We care more about the female students' feelings of and reaction to pair programming, than about their performance in the class. The objective of this research is to find a grounded theory of the effect of pair programming on female students. This section provides the parameters and methods of the research.

3.1. Research Settings

This research was during the pilot study phase of a three-year project about women in the information technology field. A study was run at NCSU in the 2003 fall semester with an undergraduate-level Software Engineering course (CSC326). There were 103 students in the class, including 16 (15.5%) females and 87 (84.5%) males. Ninety-three students, including 15 (16.1%) females and 78 (84.0%) males, agreed to participate in this research by signing an Institutional Review Board Informed Consent form. In this paper, only data from participating students are analyzed. Table 1 shows the GPA information of the students. Note that the female students had a higher overall GPA than the male students; however, their CS GPA was lower.

Table 1: Students' GPA information[5]

	GPA Average	Std. Deviation	CS GPA	Std. Deviation
Female	3.25	0.50	3.06	0.70
Male	3.17	0.54	3.23	0.67
All	3.18	0.53	3.21	0.67

The course in which the study took place was a three-credit junior-senior Software Engineering course. The students had two 50-minute lectures each week. Additionally, the students were divided to five lab sections of approximate 24 students per section, led by a student lab instructor. The students had three programming assignments, the first two of which were paired assignments. For the paired assignments, each student was assigned a partner. A different partner may be assigned for each assignment. After the three programming assignments, there was also a final project, in which the students worked in groups of between four and five students to write a plug-in for Eclipse[6], an open-source development environment for the Java programming language.

To allow for comparison for our research, there were two types of groups for the team project: solo groups and paired groups. In the solo groups, each team member was assigned a piece of the overall project, programmed alone and integrated completed work. In paired groups, the members practiced pair programming, where the team was divided into pairs, and each programming task was assigned pair-wise. The students might switch their pairing partners during the development.

To assign the students with groups of their favorable programming styles, they were asked to submit a short paper which provided their choice of programming style (solo, paired, or don't care), a short statement of their rationale for making this choice, and to list the students she or he did not want to work with. The students' preference of programming is shown in Table 2. For the most part, a student was assigned to a group of her/his programming style. More than half the students preferred to work in pairs. To allocate approximately the same number of students in both programming styles, the students who did not care for the programming style were initially assigned to solo groups. Additionally, prior academic performance was considered so that the groups could be as academically equivalent as possible. Finally, programming style assignment was adjusted to conform to the following rules:

---

[5] Two females did not have GPA information; one male did not have GPA information; three females did not have major GPA information; and three males did not have major GPA information.

[6] http://www.eclipse.org

1. A group could only have students of the same lab section. Therefore, if few students in a lab section chose a certain programming style, some students in the same lab section would need to change their programming style so that the number of students in a group could be either four or five.
2. The number of students in paired groups should be similar to that of students in solo groups.
3. The academic performance (see below) of students in paired groups should be similar to that of students in solo groups.
4. If a group had any female member, there should be at least two female members in the group.

The academic performance is an aggregated score based on the student's SAT-Math (SATM), overall GPA, and the midterm score of the course. It is evaluated with the following algorithm. In the algorithm, NSAT is normalized SATM score (SATM score divided by eight), and NGPA is normalized GPA score (GPA score multiplied by 25). The normalization is necessary so that all three scores (normalized SATM, normalized GPA, and midterm score) range from 0 to 100.

```
NSAT = SATMScore / 8
NGPA = GPA * 25
if SAT and GPA are unavailable then
      performance = MidtermScore
else if SAT is unavailable then
      performance = MidtermScore * 0.5 + NGPA * 0.5
else if GPA is unavailable then
      performance = MidtermScore * 0.5 + NSAT * 0.5
else
      performance = MidtermScore * 0.4 + (NGPA + NSAT) * 0.3
endif
```

The students were not enforced to use the assigned programming style, though. The purpose of programming style assignment was to gather students of similar preference together so that they could use their preferred style.

After the programming style of each student had been decided, the students were assigned to different groups randomly. However, their lists of "undesirable partners" were taken into consideration. Table 2 shows the result of programming style assignment.

Table 2: Programming Style Assignment

	Preferred Style			Assigned Style	
	Solo	Paired	Don't Care	Solo	Paired
Female	5	8	2	8	7
Male	16	44	18	38	40
All	21	52	20	46	47

Table 3 shows the students' academic performance scores regarding of the assigned programming styles, indicating the groups were essentially equivalent academically.

Table 3: Students' Performance Scores

Solo		Paired	
Average	Std. Deviation	Average	Std. Deviation
77.51	11.11	76.65	10.11

3.2. Limitations of Study

A number of limitations of this study need to be taken into consideration. First is the small size of subjects. Only three interviews and 15 retrospective reports of the students in the same class were analyzed. Although sample size is not a concern when generating a theory [15], the small sample size made it inappropriate to utilize statistical methods on the quantitative data so triangulated results [17] could be found. Additionally, generality cannot be shown in this study due to small sample size and sample similarity. Further study, as described in Section 5, will be conducted to confirm the findings in this research.

3.3. Research Methods

To attain accuracy and breadth of the result, both quantitative and qualitative methods were applied. Data collection and analysis approaches are explained in this section.

3.2.1. Data Collection

The information about students' academic performance is used. The information includes their GPA scores, assignment scores, and the final scores of the class.

There are two sources of qualitative data used in this study. One is the interviews with seven students, including three females, in the middle of the semester. The interviews were conducted and transcribed by Berenson and Slaten from the Department of Math Education [4]. These interviews were semi-structured. The interview protocol is provided in the Appendix A. The interview data of the female students are analyzed in this paper.

The other source is the students' project retrospectives. At the end of the semester, the students were asked to write two-page project retrospectives. The project retrospective is in the form of a questionnaire, including several yes/no questions and several open-ended ones. The questionnaire used to structure the project retrospectives is provided in Appendix B. In this paper, the retrospectives of 13 female women are analyzed.

3.2.2. Data Analysis

The focus of this paper is to form theories from the data we collected. In this research, there are large amounts of qualitative data. In the project retrospectives, some questions are more objective and concrete. The students' answers to these questions are quantified. The quantified information is not the focus of this study, but it is used with qualitative data to study the same phenomenon [17]. The quantifying coding process is not suitable for the analysis of the student interviews and the open-ended questions in the retrospectives. Grounded theory was used to analyze those less-structured data.

In addition to quantifiable data such as GPA and course grades, the students' attitudes and feelings about software development are the center of this research. The interviews are the most informative data for this purpose. The coding process applied in this research is described as follows.

Open coding. The interviews with female students consist of approximately 30 pages of textual data. In the open coding process, the categories for each interview were identified. The result of open coding is written in pairs in the form of <category, excerpt>. Each category is labeled with a short noun. An example is given as follows.

Category	Sense of dependency
Excerpt	Especially after you've done two pair programming assignments and when you go back to the old style of doing solo, it kind of feels a little bit hard. It suddenly feels like that the weight is heavier like you have to do all this stuff on your own and there's nobody to talk to and to ask a question to.

Open coding helps highlight the points in the unstructured textual data. After this process, the female students' interview was reduced to approximately five pages of more structured textual information. Additionally, the categories were clearly identified.

Axial coding. After the categories were identified via open coding, we performed additional analysis to find the interrelationships among the categories. The interrelationships were represented as cognitive maps [24]. For example, Figure 3 demonstrates the result of axial coding in the previous example.

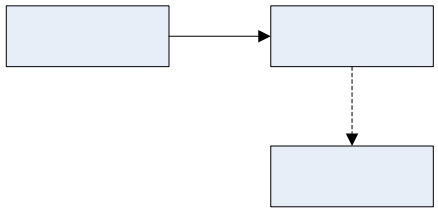

Figure 3: Categories Interrelationships

In the diagram, each block is a category. A solid arrow is a positive relationship, and a dashed arrow is a negative relationship. In this example, the diagram shows that, in the case of this student, pair programming increased her sense of dependency, which made her think solo programming was difficult.

Selective coding. The axial coding resulted in several tangible diagrams. In selective coding, the focus was on finding the categories of interest and on determining if some patterns can emerge about these categories. If such patterns can be found, we can form a theory based on such patterns. For example, from the cognitive map in Figure 3, we can form the proposition:

The student learns to depend on the partner after getting used to pair programming. However, this sense of dependency makes it difficult for the student to program alone.

Sometimes we cannot find recurring patterns. If this happens maybe some categories are missed out, or the patterns cannot form a theory. In the prior case, we need to go back to the original data to find out new categories or go back to the field to find out new evidence.

4. RESULTS

In this section, the results will be presented, grouped by the categories of interest. Three central categories are shown: enjoyment, study habits, and pair programming effectiveness. Based on the interviews and project retrospectives, the factors affecting each of the main categories will be discussed.

4.1. Enjoyment

What aspects of software development interest girls? In his classical book *The Mythical Man-Month*, Brooks lists the joys of programming [8]. They are: the joy of making things; the pleasure of making things that are useful to other people; the fascination of fashioning complex parts and watching them work in subtle cycles; the joy of learning; and the delight of working in a tractable medium. The final project of this course has all these five properties: it is about making a useful software tool; composing several software parts; learning new framework for a development environment; and working in a tractable medium. Did the female students enjoy the term project because of these properties? Are there other factors of enjoyment in a team project?

In the retrospective, the students were asked whether they enjoyed the project. Figure 4 shows the cognitive map from the female students' response. Four out of seven female students in solo groups and three out of six female students in paired groups said they enjoy the project.

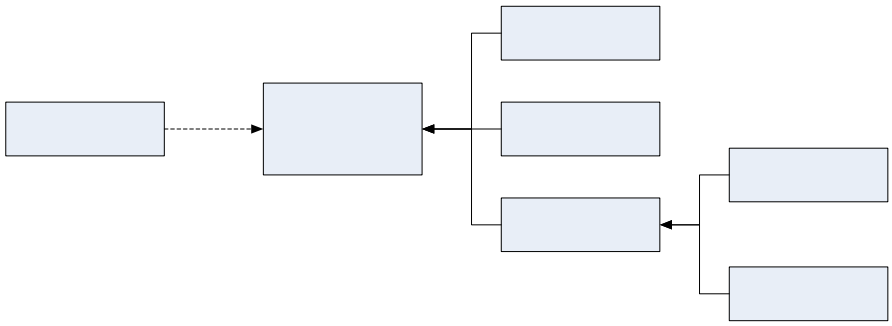

Figure 4: Project Enjoyment Cognitive Map

Although earlier studies have shown that paired students have higher level of enjoyment than solo students [21], the female students in this class did not show such trend. The three paired students who did not enjoy the project have a similar reason. They did not like the project because they need to write a plug-in for a specific platform. To write a plug-in, one needs to have thorough knowledge about the platform, which is not the material in the class. One student expressed her concern of the overwhelming efforts to learn the skill:

> *Definitely, we didn't like the fact that the project was an Eclipse plug-in because we didn't have enough time to study it well enough to avoid all that problems with Eclipse plug-in.*

Nevertheless, plug-in programming is a new idea to the students. Learning a new technique also brings the pleasure of learning. Some students, both paired and solo, enjoy the project because it is a new experience to them. For example, one student said this in the retrospective:

> *Because I've never done a plug-in before, it posed a huge challenge to me… now I feel a lot more confident in approaching a new project on a complete new topic… One thing I realized is that you don't have to be taught all the necessary knowledge before you start the project. Self-learning is very important skill.*

Some of the students enjoyed the project because this project can be useful in the real world. Compared to men, women tend to link their interest in computers to other areas that are beneficial to society [20]. Being a plug-in on a popular platform, this project is enjoyable to some female students. A student expressed her feelings about programming assignments:

> *I enjoyed the project, mainly because I felt that we were creating something that could actually be used by a customer rather than just an arbitrary program that taught us different aspects of Java like many of the programs that I've written up until now.*

Another important factor that made the student enjoy or dislike the project is people. In a team project, the interaction among the team members can make the project a complete success or total failure. While some students enjoyed the teamwork, some other students expressed two problems concerning people, including leadership (*I did not enjoy working on a team that was lead by a young lady with a rude attitude about everything*...) and unbalanced workload (…*certain people in my group did not contribute and it caused a lot of stress*.). We can see these problems as lack of skills for group work, as indicated by Waite et al. [30]. In their paper, the authors suggest to offer more open discussion, group assignments, and project courses in the course design to improve students' collaborative skills.

4.2. Study Habits

There are no explicit questions concerning students' study habits in the interview protocol or retrospective questionnaire. However, from the students' answers, we can still find out the trends of their study habits. The cognitive map that illustrates the cause and effect of the students' study habits is shown Figure 5. The interviews show that pair pressure has both positive and negative effects on the students' efficient usage of time.

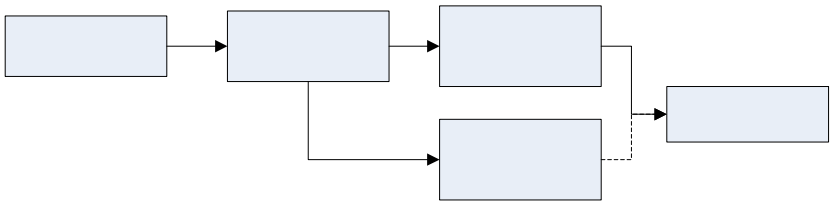

Figure 5: Pair Pressure Cognitive Map

Beyer et al. report that female students show stronger interpersonal attachment than males [6, 7]. When interpersonal attachment works in pair programming, it becomes pair pressure. Williams et al. describe pair pressure as a force that makes the programmers work smarter and harder, and draw their concentration on the programming tasks [32]. For female students, the source of this force comes from their attitude that they do not want to upset their partners, and the result is that they put more attention on their work. A student had a description of how she feels when works in pair:

> *When we meet, we'll do stuff, we'll really talk about the issues we have or really be productive as opposed to just sitting there and you don't want wasting the other person's time…*

According to the students' opinions, time management was critical for their assignments and the term project. From the retrospective essays, ten female students pointed out that time management is the most important successful factor for the term project. The effect of pair pressure on time management is two-fold. The three female interviewees stated that they felt responsible for their partners when doing the paired assignments, and this sense of responsibility helped them manage their time better. A student talked about her feelings of pair programming:

> *I think that it holds people more accountable. It makes people not do as much work at the very last minute, since they have to schedule with their partner and they just are better about planning it and working on it… overall they take more time to do it right.*

On the other hand, pair pressure can have negative effect on time management when the student needs to work alone, which is usually the case for college students. A lot of students did not get good grades in the solo assignment. Most of these students thought that assignment was particularly difficult because it was about a new concept for them (design patterns). Nevertheless, two female interviewees who did not finish the solo assignment said that they could have finished it if they had managed their time better. One of the interviewees who did not finish the solo assignment said she might have finished it if she could pair with someone:

> *… the last one we had (the third assignment)… I didn't get started on it until very late and I didn't get it finished… But I think if I had been working on it with someone else I probably would have started earlier.*

One student said the negative effect comes from the sense of dependency when they are accustomed with pair programming:

> *After you get used to pair programming, and then you have to go back to the solo programming, you will sort of feel too dependent, too relied, sort of need the other person for the job as opposed to be very independent… (Pair programming might take away the ability to work alone) not in the sense of programming ability, but the ability of time management…*

However, the negative effect of pair pressure can be avoided. For example, the teacher can design more group assignments instead of individual ones, or break a big assignment into several smaller pieces.

4.3. Pair Programming Effectiveness

The three female interviewees all agreed that pair programming is an efficient working style, no matter which programming style they preferred in the final project. They think pair programming is efficient because it can reduce debug time and help the students with problem solving. The cognitive map is shown in Figure 6.

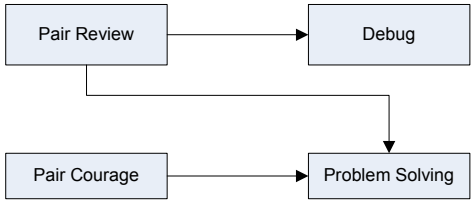

Figure 6: Pair Programming Effectiveness Cognitive Map

When programming in pairs, the driver controls the keyboard and the mouse, and explains what he or she is doing to the navigator. The explanation can help the driver concentrate on the problem. If the driver shares his or her thoughts with the navigator, the flaws in driver's reasoning can be spotted more easily. At the same time, the navigator also watches over the driver's shoulder and catches some errors that the driver is not aware of. A student shared her opinion in the interview:

> *If you're explaining your reasoning, you see flaws easier… if you do it on your own, you're probably going to go away and code the whole thing and then suddenly you realize, oops, I don't know what I was actually doing here… Literally as you're typing something at the screen, they (the partners) can correct it as you go along.*

Although traditionally programming is considered as a task that is preformed by individuals [19], the female students did not have any negative feelings when programming collaboratively. Instead, they felt more confident when working in pairs. This kind of confidence is not, however, exactly the same as described in Williams and Kessler's book [33]. In their book, the authors say pair programmers are more confident to try out solutions which they may not do when working alone. In addition to that, the female students we interviewed said they felt more comfortable to seek help from the lab assistants, other resources, or from their partners, when they work in pairs. One female student we interviewed said she felt more assured when she worked with her partner:

> *…in a situation where you don't understand some stuff and then you can ask the other person… if she didn't understand it, then you know you're not the only person… If we want to get the work done we ask the TA… If I'm on my own I probably wouldn't do that that often.*

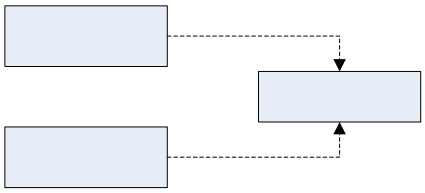

Figure 7: Negative Effects on Pair Programming

One way to examine pair programming effectiveness for the female students is to look at their grades of the final project, which is shown in Table 4. Although the female students had positive feelings about pair programming, the pair groups did not do as well as the solo groups. We can find out some clues why pair programming did not work out from the interviews and from the students' retrospective reports. The negative effects on pair programming are shown in the cognitive map in Figure 7.

Table 4: Female Students' Performance of the Final Project

	Number of Students	Average	Std. Deviation
Solo Groups	8	104.50	6.24
Pair Groups	7	96.07	11.24

In the project retrospectives, the students were asked how much time they spent solo on the final project. The students in solo groups all spent more than 50% of their time solo. Half of the students in pair groups did not answer that question, two said 25% of their

time was dedicated to solo programming, and one said 66%. All seven females in pair group said the different schedules of the team members made it difficult to practice pair programming. Unmatched schedule was the most common cause that stopped the students practicing pair programming.

Bad pairing experience is another reason why some students preferred to work alone. One of the female students we interviewed was abandoned by her partner two days before the first assignment was due. Even though pair programming worked well in her second assignment, she still chose to be in a solo group in the final project. Another interviewee said her partner (a male student) did all the work without her in one of the paired assignments, and she was *very annoyed* that she was not part of it. She believed that it was because her partner could not communicate well with her:

> *He kind of said, which I kind of understand, that he didn't want to have someone he can't explain all his thoughts to someone as he was doing it.*

It is technically difficult for the instructors to prevent schedule mismatch and bad pairing experience. It is virtually impossible to have a grouping scheme that resolves the schedule mismatch issue, since every student has different classes. It is also impractical for the instructors to monitor all the pairing sessions to make sure that everything goes well. Radical solutions, such as distributed pair programming [2], may be needed to solve these issues.

5. CONCLUSION AND FUTURE WORK

After thorough analysis of the qualitative data from 15 female students of an undergraduate level Software Engineering course, including three interviews and their retrospective essays, we find out the following phenomena:

- The project enjoyment comes from the usefulness of the project and teamwork.
- Pair programming helps the female students with time management because they think they are responsible for their partners.
- When the female students are used to pair programming, it can be difficult for them to get back to solo programming.
- Pair programming helps the female students work more efficiently in programming tasks.
- Schedule mismatch and bad pairing experience is the enemy of effective pairing programming.

This research explores the effects of pair programming on female students, and also brings up the difficulties to use pair programming in a classroom. From the result, we find out the following problems that need to be addressed in future study:

- How do we improve the students' collaborative skills?
- How do we avoid the negative effect of pair programming, i.e. sense of dependency?
- How do we resolve students' mismatched schedules?
- How do we make sure the workload equity in group assignments?

We are conducting more qualitative and quantitative inquiries to make the findings more reliable. From the qualitative perspective, more observations and interviews will provide

more supportive or contrary evidences, and richer context of the phenomena. Quantitative experiments are needed to test the generality of the findings.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 00305917. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] AAUW Educational Foundation: *Tech-Savvy: Educating Girls in the New Computer Age*, American Association of University Women, 2000.

[2] Prashant Baheti, Edward F. Gehringer, and P. David Scotts: "Exploring the Efficacy of Distributed Pair Programming," *XP/Agile Universe 2002*, pp. 208 – 220, 2002.

[3] Kent Beck: *Extreme Programming Explained: Embrace Change,* Addison-Wesley, 2000

[4] Sarah Berenson, Kelli M. Slaten, and Laurie Williams: "Collaboration through Agile Software Development Practices: Student Interviews and Lab Observations," North Carolina State University Technical Report, NCSU CSC TR 2004-12, 2004.

[5] Sylvia Beyer: "The Accuracy of Academic Gender Stereotypes," *Sex Roles*, Vol. 40 Nos 9/10, pp 787 – 813, 1999.

[6] Sylvia Beyer, Kristina Rynes, and Susan Haller: "Deterrents to Women Taking Computer Science Courses," *IEEE Technology and Society Magazine*, Vol. 23 Issue 1, pp 21 – 28, 2004

[7] Sylvia Beyer, Kristina Rynes, Julie Perrault, Kelly Hay, and Susan Haller: "Gender Differences in Computer Science Students," *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, pp 49 – 53, 2003.

[8] Frederick P. Brooks, Jr.: *The Mythical Man-Month*, Addison-Wesley, 1995.

[9] Tracy Camp: "The Incredible Shrinking Pipeline," Communications of the ACM, Vol. 40 No 10, pp 103 – 110, 1997.

[10] Alistair Cockburn and Laurie Williams: "The Costs and Benefits of Pair Programming," in Extreme Programming Examined, Addison-Wesley, 2001.

[11] Joann McGrath Cohoon: "Toward Improving Female Retention in the Computer Science Major," *Communications of the ACM*, Vol. 44 No 5, pp 108 – 114, 2001.

[12] Joann McGrath Cohoon: "Must There Be So Few? Including Women in CS," *Proceedings of the 25th International Conference on Software Engineering*, pp 668 – 674, 2003

[13] John W. Creswell: *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches, 2nd Edition*, Sage Publication, 2003.

[14] Norman Denzin and Yvonna Lincoln (editors): *Handbook of Qualitative Research, 2nd Edition*, Sage Publications, 2000.

[15] G. Glaser and Anselm L. Strauss: *The Discovery of Grounded Theory: Strategies for Qualitative Research*, Aldine de Gruyter, 1967.

[16] Andrea Jepson and Teri Peri: "Priming the Pipeline," *SIGCSE Bulletin*, Vol. 34 No 2, pp 36 – 39, 2002.

[17] Todd D. Jick: "Mixing Qualitative and Quantitative Methods: Triangulation in Action," *Administrative Science Quarterly*, Vol. 24, pp 602 – 611, 1979.

[18] Neha Katira, Laurie Williams, Eric Wiebe, Carol Miller, Suzanne Balik, and Ed Gehringer: "On Understanding Compatibility of Student Pair Programmers," *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, pp 7 – 11, 2004.

[19] Jane Margolis and Allan Fisher: "Geek Mythology and Attracting Undergraduate Women to Computer Science," *Impacting Change Through Collaboration, Proceedings of the Joint National Conference in Engineering Program Advocates Network and the National Association of Minority Engineering Program Administrators*, 1997.

[20] Jane Margolis, Allan Fisher, and Faye Miller: "Caring about Connections: Gender and Computing," *IEEE Technology and Society Magazine*, Vol. 18 Issue 4, pp 13 – 20, 1999.

[21] Charlie McDowell, Linda Werner, Heather E. Bullock, and Julian Fernald: "The Impact of Pair Programming on Student Performances, Perception, and Persistence," *Proceedings of the 25th International Conference on Software Engineering*, pp 602 – 607, 2003.

[22] Nachi Nagappan, Laurie Williams, Miriam Ferzli, Eric Wiebe, Kay Yang, Carol Miller, and Suzann Balik: "Improving the CS1 Experience with Pair Programming," *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, pp 359 – 362, 2003.

[23] National Center for Education Statistics: *Digest of Education Statistics, 1990-2002*, Institute of Education Sciences, U.S. Department of Education, 1990-2002.

[24] Gery W. Ryan and H. Russell Bernard: "Data Management and Analysis," in *Handbook of Qualitative Research, 2nd Edition*, Sage Publications, 2000.

[25] Dean Sanders: "Student Perceptions of the Suitability of Extreme and Pair Programming," Proceedings of 2001 XP Agile Universe, 2001.

[26] Carolyn B. Seaman: "Qualitative Methods in Empirical Studies of Software Engineering," *IEEE Transactions on Software Engineering*, Vol. 25 No 4, pp 557 – 572, 1999.

[27] Anselm L. Strauss and Juliet M. Corbin: *Basics of Qualitative Research: Techniques and Procedures of Developing Grounded Theory 2nd Edition*, Sage Publications, 1998.

[28] Lynda Thomas, Mark Ratcliffe, and Ann Robertson: "Code Warriors and Code-a-Phobes: A Study in Attitude and Pair Programming," *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, pp 363-367, 2003.

[29] Moshe Y. Vardi, Tim Finin, and Tom Henderson: "2001 – 2002 Taulbee Survey: Survey Results Show Better Balance in Supply and Demand," *Computer Research News*, Vol. 5 No 2, pp 6-13, 2003. Also available at http://www.cra.org/CRN/articles/march03/taulbee.html.

[30] William M. Waite, Michele H. Jackson, and Paul M. Leonardi: "*Student Culture vs Group Work in Computer Science*," *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, pp 12 – 16, 2004.

[31] Laurie Williams and Robert R. Kessler: "The effects of "pair-pressure" and "pair-learning" on software engineering education," *Proceedings of 13th Conference on Software*

Engineering Education and Training, pp 59 – 65, 2000.

[32] Laurie Williams, Robert R. Kessler, Ward Cunningham, and Ron Jeffries: "Strengthening the Case for Pair-Programming," *IEEE Software*, Vol. 17 Issue 4, pp 19-25, 2000.

[33] Laurie Williams and Robert R. Kessler: *Pair Programming Illuminated*, Addison-Wesley, 2002.

[34] Laurie Williams, Kai Yang, Eric Wiebe, Miriam Ferzli, and Carol Miller: "Pair Programming in an Introductory Computer Science Course: Initial Results and Recommendations," presented at the 17th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2002), 2002.

Appendix A. Interview Protocol: Mid-semester Pilot, October 2003

OPENING:

SAY: We are evaluating the instructional approaches used in your computer science 326 course. It is important for the instructors to know how well these new methods are working for you. This is why we want to ask some of the students what they think about the programming assignments in this course.

Your responses will be anonymous. While we will share the results of the interviews with the instructor, he or she will not know that you were interviewed.

1. What do you think of the assignments so far?
    a. Can you explain why you think that?
    b. Can you give me an example?

2. So some of the assignments have been paired and some have been solo. What do you think of the students in your lab prefer, pair or solo?
    a. What reasons do they give for preferring _____?
    b. What about those who prefer _____?

3. What about you? What approach do you prefer?
    a. Why is that true?
    b. Any other reasons?

4. What makes _____ an effective instructional tool for you?
    a. Have you noticed other approaches that help you learn?
    b. Can you give me an example?

5. What do you see yourself doing after graduation?
    a. Have you ever done this before? How did it go?
    b. How do you envision the workplace?

6. Do you think pair programming will work in today's IT workplace?
    a. Why? Please explain this idea some more.
    b. Why not? Please explain this idea some more.

Thanks very much for your time. You were very thoughtful. Your ideas will help many students here at NC State and in other programs across the United States. Good luck with the semester.

Appendix B. Project Retrospective Questionaire

1. User Stories, Iteration, Acceptance Tests: Did having three short iterations help you pace yourself to completion? Do you feel you benefited from feedback you got after each iteration? Did having the Acceptance Tests help you to clarify the requirements?
2. Configuration management: Did CVS help you manage your files?
3. JUnit and FIT: Do you feel either or both of these kinds of testing helped you in creating a high quality project? Do you feel like this kind of testing helped reduce debugging time? Did you create these test cases as you went along or did you complete a user story and then run the tests? Based on your experience, do you think it is best to create the tests all along or at the end?
4. Your method of work: Pair programming or Solo programming. How did it work? What percentage of time do you feel like you worked solo, what percentage of the time do you feel you worked in pairs? What made you decide when to work solo and when to work in pairs [time constraints, difficulty of work, etc.]
5. How well your team communicated with each other. What vehicles did you use for communication – web site, email, setting up a mailing list, etc?
6. Division of team roles – both the team roles discussed in class (team leader, development manager, quality manager. planning/process manager) . . . but also did you assign a certain person to be the SWT expert, etc. How was technical information shared within the team?
7. Did you enjoy the project? Why or why not? Did you like the fact that it was an Eclipse plug-in? [This question will help immensely for choosing next year's project.]
8. Did you reuse any code found on the web? How did this go? How was testing this code?
9. How did you like Extreme Programming? Do you think your project would have instead been better run in a plan-driven way? Did you formally or informally create use cases, class diagrams, or sequence diagrams? If so, did you share them among the team? Did you run any CRC card sessions?
10. What advice do you have for next year's class that will have a 4-5 person, 6-week team project [which may or may not be an Eclipse plug-in?] [Mark this "you can share" or "please do not share." I will create a web page with snippets of advice if that's OK with you.]
11. Did you have any particularly stubborn defects? Can you analyze the cause of these defects – is there anything you'd change about what you did to prevent such defects?
12. Analyze the effect of keeping Sev 1 and Sev 2 defects out of your project. Do you think you ended up with a better structured code base because of them?
13. Please include anything else you'd like to share.