ABSTRACT

HAYWARD, LAUREN JANECE. Students' Perceptions of Distributed Pair Programming in an Upper-Level Undergraduate Software Engineering Course. (Under the direction of Dr. Laurie Williams).

Pair programming is a style of programming in which two programmers work together on the same software artifact. Pair programming has many demonstrated benefits including higher code quality and enhanced personal enjoyment. Unfortunately, students do not always find meeting together in the same physical location to pair program to be convenient or even possible. In this case, they may choose to pair program from distributed locations.

To examine student perceptions about distributed pair programming, we conducted three studies with undergraduate software engineering students over two semesters. In our first study, all 49 students enrolled in the course had the option of using two distributed pair programming tools. We then analyzed their responses to an end-of-course retrospective that included questions about pair programming. We also analyzed their responses to a pair programming extra credit option. In our second study, we interviewed students from the same semester to more closely examine their perceptions about distributed pair programming. In our third study, we surveyed students the following semester about the benefits and drawbacks of distributed pair programming before and after trying distributed pair programming in a structured setting for 90 minutes.

We found that students who pair program remotely perceive that they enjoy many of the same benefits as those who pair program while collocated, including higher code quality

and personal enjoyment. Students are motivated to pair program remotely because of the feedback they receive from their partners, convenience and comfort of working from home, personal responsibilities that require them to work from home, and decreased scheduling issues of distributed pair programming compared to collocated pair programming. The main reason students described for not pair programming remotely was technical difficulties with the distributed pair programming tools. Students reported less productivity initially when pair programming remotely due to the learning curve of the tools. Students also listed scheduling conflicts and a preference for collocated work as other reasons for not pair programming remotely.

Based on the results from the three studies described in this work, we contend that students in upper-level programming courses could find distributed pair programming beneficial as long as the teaching staff provides technical support for the distributed pair programming tool. To help alleviate the technical issues that may discourage students, we recommend giving students class time to learn how to install and use the distributed pair programming tool. Because of students' feedback in these studies, and prior work by Chong and Hurlbutt as well as Hanks, we feel using a distributed pair programming tool without explicit driver and navigator roles that allows for gesturing and the sharing of more than one file is beneficial.

Students' Perceptions of Distributed Pair Programming in an Upper-Level Undergraduate Software Engineering Course

by
Lauren J. Hayward

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Master of Science

Computer Science

Raleigh, North Carolina

2009

APPROVED BY:

_____	_____
Dr. Laurie Williams Committee Chair	Dr. Ed Gehringer
_____	
Dr. Mladen Vouk	

DEDICATION

To Stephen Colbert

BIOGRAPHY

Lauren Hayward was born in New Bern, North Carolina. As a child, she moved to Arizona, Alabama, and Nebraska, finally returning to New Bern where she graduated from high school in 2004. She received her Bachelor of Science in Computer Science from North Carolina State University in 2007. After graduation, Lauren will work as a software engineer at IBM in Research Triangle Park.

ACKNOWLEDGMENTS

To Jason. Thank you for your love. We are a perfect pair!

To Mom and Dad. Thank you for you always loving and supporting me. Without your encouragement, I would not have entered computer science.

To Deanna. You always make me smile. I am finally finishing college, and you are just beginning. Have fun in college, little sister!

To Mom and Dad Schaefer. Thank you for all of the home-cooked meals and moving me multiple times over the last year. You enabled me to focus on my school work when the bed bugs bit.

To Laurie. Thank you for all of your assistance and guidance. I would not be graduating without your help. Most importantly, thank you for showing me that computer scientists can wear fabulous shoes!

To my committee. Thank you for your insightful feedback.

To the Realsearch group. Thank you for your friendship. I have learned so much from all of you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

Pair programming is a practice with many demonstrated benefits including higher code quality [6, 27, 28], enhanced technical skills of the programmers [6, 27], improved team communication [6, 27, 28], enhanced personal enjoyment [6, 27, 28], and "pair pressure" where partners help keep each other on task [22, 27, 28, 33]. Traditionally, the programmers sit side-by-side at the same computer [27].

Unfortunately, students may not always find meeting together in the same physical location convenient or even possible. A student may not be able to meet in person due to schedule conflicts or family commitments [12]. Lab space with the necessary software may not be available. Students may desire to work from their dorm rooms, or students may need help debugging an unforeseen issue while working at home. Distance education courses are on the rise [24], and students in computer science distance education classes often need a way to complete group projects with their distributed teammates. In these cases, students may benefit from pair programming from distributed locations.

If the technology is available, will students pair program remotely? If so, will they find it beneficial? *The goal of this research is to examine student perceptions about the benefits and drawbacks of distributed pair programming*. In this work, we use distributed or remote pair programming to refer to a style of programming in which two programmers who are geographically-distributed and synchronously collaborating over the Internet work together on the same software artifact. We use collocated pair programming to refer to a style of programming in which two programmers who are physically side-by-side work

together on the same software artifact. We use pair programming to refer to the union of distributed pair programming and collocated pair programming.

The following five research hypotheses and two questions were examined:

H1a: *Students who pair program remotely will enjoy many of the same benefits as those who pair program while collocated, including higher code quality and personal enjoyment.*

H1b: *Women and students from other underrepresented groups in computing who pair program remotely will enjoy many of the same benefits as those who pair program while collocated, including personal enjoyment.*

H2: *Distributed pair programming decreases the scheduling issues that arise for students trying to schedule collocated pair programming.*

H3: *Making distributed pair programming technology available to students increases the likelihood that they will pair program.*

H4: *Trying distributed pair programming increases the likelihood that students will pair program remotely in the future.*

RQ1: *What motivates students to or not to pair program remotely?*

RQ2: *What do students perceive as the benefits and drawbacks of distributed pair programming?*

We consider a hypothesis to be *supported* if all evidence we found supports the hypothesis, *partially supported* if part but not all of the evidence we found supports the hypothesis, and *not supported* if none of the evidence we found supports the hypothesis.

2 About the Studies

We conducted three studies with undergraduate software engineering students at North Carolina State University (NCSU) to examine the research hypotheses and questions presented in Chapter 1. The first two studies involved students in the fall semester of 2008, and the third study involved students in the spring semester of 2009.

2.1 About the Software Engineering Course

Early in each semester, the software engineering teaching staff introduces techniques for successful pair programming[1] to students. The course has a two-hour weekly lab held in the Laboratory for Collaborative System Development (LCSD)[2]. The laboratory has a pair programming setup where each computer has two monitors, keyboards, and mice so that partners can easily work together without crowding around the same monitor. Students have no choice during the weekly lab but to pair (and "bond") with their partner. Students can choose to obtain key fobs so they can access the LCSD at any time. Students in this course use Rational Team Concert[3], which is built on Eclipse[4], as their integrated development environment (IDE).

Students in this course have two, two-week paired assignments where they fix bugs in and implement new requirements for a web application. Students are encouraged to pair program as they complete these paired assignments. After the paired assignments, students

[1] Students were shown the "Fun with Pair Programming" video available at http://agile.csc.ncsu.edu/pairlearning.
[2] http://collaboration.csc.ncsu.edu/laurie/LCSD.htm
[3] http://jazz.net/
[4] http://www.eclipse.org/

have a solo two-week assignment. Finally, students have a three, four, or five person team project that they have five weeks to complete. Students have a portion of the weekly labs to work on their assignments or team project but they typically must work for approximately four more hours a week to complete the assignments.

Throughout the course, students use PairEval[5], a web-based peer evaluation tool, to document information about their personalities and evaluate their partners. As part of their first homework assignments, students take a Myers Briggs personality test[6] and input the results in PairEval. They also rate their own work ethic on the following scale 1 (just barely get by) to 9 (get the best grade you possibly can) and record it in PairEval. The teaching staff assigns partners for assignments based on students' Myers Briggs Type Indicator [14] results and self-reported work ethic. Students with a difference in their sensing-intuition scale from their Myers Briggs Type Indicator are predicted to be compatible, and students with similar self-reported work ethic are also predicted to be compatible [29]. The teaching staff can use PairEval to create teams. For example, Figure 1 shows a teaching assistant placing Jiang and Yonghee in the same team. Students can then login to PairEval and evaluate their partners.

[5] http://agile.csc.ncsu.edu/paireval
[6] http://www.humanmetrics.com/cgi-win/JTypes2.asp

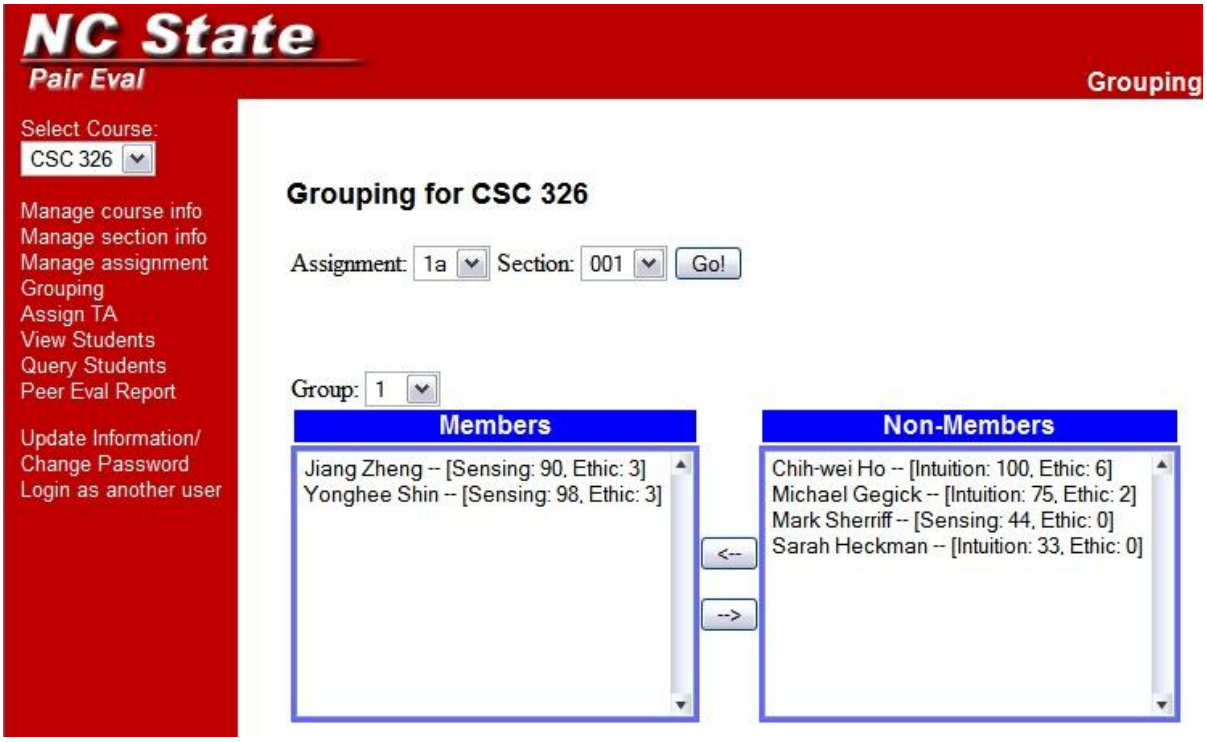

Figure 1. Creating teams in PairEval.

2.2 Data Analysis

Student responses from the three studies presented in this work were quantitatively and qualitatively analyzed based on methods presented in [21]. All quantitative data was input directly into a spreadsheet for analysis. To code the qualitative data, each question was assigned a code. Then a researcher read through the student responses and labeled responses to the questions with subcodes. For example, two of the subcodes for the code "What made you decide to work in pairs?" were "Discuss implementation direction" and "Debugging code." The subcodes were then entered into a spreadsheet organized by student id and code. Throughout the coding process, subcodes were added, deleted, merged, subdivided, and modified as necessary. To ensure that subcodes were assigned accurately, two passes were

made over each student response. Finally, a researcher analyzed the quantitative data, codes, and subcodes in the spreadsheet to find patterns and trends.

3 Background

In this chapter, we discuss pair programming, the need for distributed pair programming, and tools that facilitate distributed pair programming.

3.1 What is Pair Programming?

Pair programming is a style of programming in which two programmers work together, continuously collaborating on the same design, algorithm, code, or test. When pair programming, one partner serves as the driver who actively writes the design, algorithm, code, or test. The other partner, the navigator, observes the driver's work to look for errors and thinks strategically about what should be done next. Both partners are constant brainstorming partners. Traditionally, the programmers sit at the same computer and periodically switch roles [27].

While the name "pair programming" may imply pairing only during the implementation portion of the process while the partners are writing code, partners can pair during all phases of development including design, implementation and test. In fact, the more complex the task, the more beneficial having a partner becomes [27]. Pair programming is an agile [5] software development methodology that has gained acceptance among agile and plan-driven teams and also in education.

To keep both partners engaged, rotating the roles of driver and navigator is considered important [23, 27, 32]. However, in a four-month ethnographic study [4], two professional software development teams practicing pair programming did not follow the strict driver and navigator roles. Rather, when the partners had equivalent expertise, they

engaged jointly in discussion and brainstorm. In this case, the driver mainly served the role of typist. Programmers appeared to be most effective when they jointly took on the roles of driver and navigator. Programmers seemed more engaged when they were in possession of the keyboard or perceived keyboard control was imminent. Analyses performed by Höfer [13] and Freudenberg et al. [9] support the findings of Chong and Hurlbutt.

3.2 What Does Effective Pair Programming Look Like?

Researchers have found several best practices for effective pair programming, and we highlight six of them here:

- **Periodically switch who is driving**. By switching who is driving fairly regularly, both partners remain engaged in the pair programming session. At any time, the navigator should be able to jump in and begin coding [23, 27, 32].
- **Leave your insecurities at the door**. Pair programmers who work with partners with insecurities about their programming skills find it difficult to be successful. Programmers should use the pair programming session as an opportunity to become better programmers rather than dwelling on their insecurities [27, 32].
- **Be humble.** Even the most experienced pair programmers can learn something from less experienced programmers while pair programming. When the more experienced programmer comes into the session with a positive attitude, the less experienced programmer can ask the "why" questions, which may lead to a better solution [27, 32].

- **Take a break!** Pair programming can be fairly intense as partners typically keep each other on task. Partners should take breaks during the session so that they can come back to the session with a fresh mind [27, 32].
- **Resist competing with your partner.** Pair programming partners should work together as equals to complete the tasks. They should not blame each other for defects or elevate one partner as the reason why they succeeded [27, 32].
- **Communicate constantly.** Effective pair programmers chat constantly so that the driver is articulating what he is doing (which can sometimes even help the driver catch his own errors) and the navigator remains aware of what the driver is doing [27].

3.3 Why Pair Program?

While it may seem a bit redundant at first to ask two programmers to work together, researchers have reported several benefits of working in pairs:

- **Equivalent time spent**. Pairs complete their work with only about a 15% cost (rather than 100% as might be expected) when they work together, and many pairs report completing their tasks in about half the time as they would working individually [6, 27, 28].
- **Higher code quality**. Pairs typically produce higher code quality with fewer defects than individuals working alone. Because defects are found earlier in the process, they are much less expensive to fix [6, 27, 28].

- **Enhanced technical skills**. Those who pair program tend to enhance their technical skills as they work with their partners. They report learning from their partners as they pair program. Pair programmers may also introduce parts of the system to their partners that their partners may have not otherwise seen [6, 27, 28].
- **Improved team communication**. Teams who make use of pair programming have reported an improved level of team communication. As team members pair program, they get to know each others' strengths and weaknesses. When taking breaks during pair programming sessions, partners build informal relationships, which help improve communication [6, 27, 28].
- **Enhanced enjoyment**. Pair programmers report that they enjoy their work more when working with a partner [6, 27, 28].
- **Pair pressure**. Pair programmers have reported a feeling of "pair pressure" that positively influences their work. They set aside specific times to work with their partners rather than waiting until the last minute to begin as they may have if they were working alone. Additionally, since their partners are watching, programmers are less likely to waste time browsing their email or surfing the Internet [22, 27, 28, 33].

3.4 Why Work in a Distributed Way?

A need for distributed work has arisen in recent years in both the academic and industrial realms.

Distance education classes are on the rise. In the 2000-2001 school year, 56% of all 2-year and 4-year Title IV-eligible, degree-granting institutions offered distance education courses, and 12% of all institutions reported that they intended to begin offering distance education courses over the next three years [24]. Students in computer science distance education classes often need a way to complete group projects with their distributed teammates. Additionally, students in traditional classes may need to work remotely. A student may not be able to meet in person due to schedule conflicts or family commitments [12]. Lab space with the necessary software may not be available. Students may desire to work from their dorm rooms, or students may need help debugging an unforeseen issue while working at home.

Similarly, teams in industry have begun to work in a distributed way for many reasons including engineers' desires to not relocate, engineers being based at separate locations, high travel costs, and a lack of office space [8].

3.5 Tools for Facilitating Distributed Pair Programming

If distributed pair programmers are to achieve the same benefits as collocated pair programmers, they need tools to support cross-workspace visual, manual, and audio channels [7]. These channels enable partners to collaborate through sometimes subtle ways. For example, a small headshake or a mumble could serve as communication between the partners. Distributed pair programmers may want to use a video feed and/or voice feed to help facilitate communication while pairing.

When determining which distributed pair programming tool to use, several features should be considered:

- **Does the tool easily interface with the current integrated development environment (IDE) being used?** Desktop sharing tools like Elluminate, LogMeIn, and VNC4DPP work with any IDE. Other tools like COPPER have their own source code editors, so programmers would need to switch to using COPPER when pair programming remotely. Eclipse plugins like DocShare, Sangam, and XPairtise easily interface with Eclipse.
- **Does the tool share the necessary files and applications?** The tool should support sharing the type of files, applications, and/or testing tools that the programmers will need.
- **Does the tool support gesturing?** Gesturing features allow partners to point to or highlight portions of the screen, much as partners would gesture to the screen if they were at the same computer. Gesturing features can be useful to distributed pair programmers [11].
- **Does the tool have explicit driver and navigator roles?** Pair programmers do not always follow the strict "driver" and "navigator" roles, so tools for distributed pair programming should allow for both partners to easily gain access to control the keyboard [4]. Many tools require users to indicate who is acting as the driver and who is acting as the navigator, which could inhibit programmers who do not want to strictly follow these roles.
- **Does the tool preserve each partner's privacy?** Desktop sharing tools allow the guest full access to the owner's computer, allowing the guest to potentially invade the owner's privacy. Applications that only allow for the sharing of specific files or applications help preserve programmers' privacy.

- **Does the tool require too much bandwidth?** Some tools transfer only the edits in the code that a partner makes, whereas other tools transfer an image of each partner's screen. Transferring screen images typically requires more bandwidth and can result in delays.

Table 1 presents a summary of eight tools that facilitate distributed pair programming.

Table 1. Tools that facilitate distributed pair programming.

Tool Name	Standalone tool?	What is shared?	Gesturing feature?	Explicit driver & navigator roles?
COPPER [18]	Yes, standalone source code editor.	COPPER editor.	Unknown.	Yes.
DocShare[7]	No, Eclipse plugin.	Eclipse editor.	No.	No.
Elluminate®[8]	Yes.	Owner's desktop or specific applications.	Yes, with a shared cursor.	No.
LogMeIn®[9]	Yes.	Owner's desktop.	Owner can gesture with cursor but guest cannot.	No.
Sangam [12, 19]	No, Eclipse plugin.	Eclipse editors, launches, resources, & refactoring.	Somewhat through text highlighting but no gesture tool.	Yes.
Sangam with Facetop [19]	No, Eclipse plugin.	Eclipse editors, launches, resources, & refactoring.	Yes, through a video overlay of the partner on the screen.	Yes.
VNC4DPP[10, 11]	Yes, built on Virtual Network Computing.	Owner's desktop.	Yes, through a gesture tool.	No.
XPairtise[10]	No, Eclipse plugin.	Eclipse projects and whiteboard.	Yes, through text highlighting but no gesture tool.	Yes.

[7] http://www.eclipse.org/ecf/
[8] http://www.elluminate.com/
[9] http://www.logmein.com/
[10] http://xpairtise.sourceforge.net/

4 Related Work

In this chapter, we describe related work on underrepresented groups in computing as well as distributed pair programming.

4.1 Underrepresented Groups in Computing

Diversity, which could include differences in gender or race, can lead to broader perspectives, greater creativity, and higher quality team performance [15]. However, computer science educational programs have largely underrepresented groups. In the 2006-2007 academic year, only 12.2% of bachelor's degree recipients in computer science were women [1]. Additionally, in the 2006-2007 academic school year, the ethnicity of bachelor's degree recipients was the following: 6.5% Nonresident Aliens; 3.4% African American, Non-Hispanic; .4% Native American/Alaskan Native; 14.6% Asian/Pacific Islander; 5.4% Hispanic, 67.3% Caucasian, Non-Hispanic; and 2.5% other/not listed [1]. Much effort has been put into increasing the recruitment and retention of women and African Americans in computing, and pair programming has been shown to help in this area [3, 17, 26, 30].

Women face many challenges as they pursue computing in college. Women often do not share the same values as the men around them and face dispiriting experiences at the university [16]. In *Unlocking the Clubhouse*, Margolis and Fisher state that "many [women] end up doubting their basic intelligence and their fitness to pursue computing." Margolis and Fisher also found that women who choose to major in computing describe a desire for "computing with a purpose." Unlike men who commonly enjoy computing for the sake of

computing, women desire for their work in computing to make a positive impact on society [16].

Walton and Cohen define *belonging uncertainty* as "in academic and professional settings, members of socially stigmatized groups are more uncertain of the quality of their social bonds and thus more sensitive to issues of social belonging" [25]. In their study, computer science students completed an activity that might lead them to believe they have few friends in the field. Caucasian students' sense of belonging and potential was unaffected by the activity, but the African American students' sense of belonging and potential were negatively affected [25]. Even though women are underrepresented in computing, women did not have a statistically significant response to the activity. Walton and Cohen [25] suggest that unlike racial stereotypes, gender stereotypes in computing focus on women's quantitative ability rather than their social belonging. The authors performed a similar study where participants completed an activity that might lead them to believe they have few skills in computing. Men's sense of belonging was unaffected by the activity, but women's sense of belonging and potential were negatively affected [25].

4.2 Distributed Pair Programming

Distributed pair programming is a style of programming in which two programmers who are geographically-distributed and synchronously collaborating over the Internet work together on the same software artifact. In this section, research results from other studies on distributed pair programming are presented.

In an experiment in the graduate-level class, Object-Oriented Languages and Systems, with 132 students at NCSU, distance education students and traditional classroom

students were assigned a team project for teams of size two to four. The students were able to choose their own teams (which could include both distance learning students and traditional classroom students) and one of four ways they wanted to work together: collocated team without pairs, collocated team with pairs, distributed team without pairs, and distributed team with pairs. The distributed pairs used headsets and microphones along with instant messaging to communicate with each other. They were able to view each other's screens by using desktop sharing software such as NetMeeting, PCAnywhere, or VNC, so the navigator had read-only access while the driver edited the code. Pairs were also provided with webcams but they chose not to use them [2].

Baheti et al. [2] measured productivity (lines of code per hour) and quality (team project grade). The researchers did not find statistically significant differences in productivity between distributed and collocated teams, though the distributed teams with pairs earned the highest project scores. Through student surveys, students on the distributed pair programming teams indicated that distributed pair programming helped increase teamwork and communication [2].

Stotts et al. [22] studied eight graduate students from two universities who worked in distributed teams of two to complete a programming project. Every team had a student from each university, and the students never met face-to-face. Two of the teams pair programmed remotely while the other two teams divided the work between the programmers and programmed alone. The researchers ran a set of 15 test cases against all four teams' projects. Both of the distributed pair programming teams passed all 15 test cases. One of the teams who programmed separately passed 12 test cases and the other team who programmed

separately did not complete the project. The distributed pair programming teams had an average shorter development time [22].

Stotts et al. [22] produced a list of lessons they learned from their studies. They suggest that distributed pair programmers meet face-to-face when possible, which helps build informal relationships between the programmers. If meeting face to face is not possible, programmers should make an effort to get to know each other on a more personal level, even if it is just through the sharing of a personal webpage. Distributed pair programmers need to talk with each other almost constantly so the navigator knows what the driver is doing. Distributed pair programmers may have an advantage over most collocated pair programmers as they will each have their own monitors, increasing their ability to easily see the code. Additionally, the navigator also has the ability to search the Internet for resources while the driver is coding. The navigator needs to remain disciplined so he does not get off task. Distributed pair programmers are forced to keep electronic copies of their work (for example, design diagrams and notes) that may otherwise get lost. Distributed pair programming has an additional learning curve beyond the collocated pair programming jelling period as programmers learn to use the tool and adjust to working remotely. Also, distributed pair programmers may have to take more time to verbally explain a concept or idea that collocated pair programmers could simply sketch on a piece of paper [22].

In a study by Ho et al. [12], four students from NCSU and four independent programmers in California worked together to develop an Eclipse plugin. They described some interesting anecdotes of working together. First, they noted that the navigator could easily become distracted during distributed pair programming because the driver was not

sitting beside him—his voice was instead coming out of the computer's speakers. To eliminate this problem, when the driver noticed the navigator talking less, the driver would prompt the navigator to contribute. Second, they noticed that the distributed pair programmers did not form the informal relationships that the collocated pair programmers did. While none of the programmers knew each other before the study, the collocated pair programmers quickly became friends; however, the programmers who did not work together from the same location had a sense of unfamiliarity between them. They did not feel this unfamiliarity prevented them from working well together.

Hanks [11] studied students in four introductory programming classes at the University of California, Santa Cruz. Students had homework partners and had the option of dividing the work or pair programming to complete the homework assignments. Students were placed in either the control group (who did not have the option of using the distributed pair programming tool VNC4DPP [10, 11]) or the treatment group (who had the option of using VNC4DPP). Hanks found no statistically significant difference on students' assignment grades or final exam scores between the groups. Students who used VNC4DPP spent less time working alone than the students who did not have the option of using VNC4DPP. Hanks' work suggests students can successfully pair program remotely.

While distributed pair programming has been shown to be better than distributed non-pair programming, distributed pair programming is not perfect. Olson and Olson argue that distance does matter; it is difficult to establish common ground and trust when working remotely. They suggest using collocated teams whenever possible, and if not, an effort should be made to have the team members meet face-to-face for team building activities [20].

5 Study 1: Student Perceptions about the Availability of Distributed Pair Programming

The goal of our first study is to examine student perceptions about distributed pair programming when a distributed pair programming tool is available for use.

We conducted a study with all 49 undergraduate software engineering students in the fall semester of 2008. We gave students instructions on how to install and use Sangam [12, 19] and DocShare[11], two Eclipse plugins for distributed pair programming. All students submitted a retrospective with answers to questions about pair programming at the end of the semester. Students who chose to pair program during their team project answered additional questions about pair programming.

Through their retrospectives, the following research hypotheses and question were examined:

H1a: *Students who pair program remotely will enjoy many of the same benefits as those who pair program while collocated, including higher code quality and personal enjoyment.*

H1b: *Women and students from other underrepresented groups in computing who pair program remotely will enjoy many of the same benefits as those who pair program while collocated, including personal enjoyment.*

H3: *Making distributed pair programming technology available to students increases the likelihood that they will pair program.*

RQ1: *What motivates students to or not to pair program remotely?*

[11] http://www.eclipse.org/ecf/

5.1 Research Methodology

We conducted a study[12] with all 49 students in an undergraduate software engineering course at NCSU. Forty-three of the students were male and six of the students were female.

During the first paired assignment, students are often overwhelmed as they learn new technology and languages in addition to learning the code base with which they will be working. To make it easier for students to adjust to this new technology, we did not introduce the distributed pair programming technology until the beginning of the second paired assignment.

During the lab session at the beginning of the second paired assignment, students were instructed to follow a tutorial that showed them how to install and use Sangam, a distributed pair programming tool. We chose to recommend Sangam because it is an Eclipse plugin that easily interfaces with Rational Team Concert, and it preserves students' privacy (unlike desktop sharing tools). The teaching staff encouraged students to use the tool to pair program remotely when they were unable to meet in the same location.

After showing students Sangam, we discovered another tool, DocShare, that we felt students would enjoy more. DocShare does not have explicit driver and navigator roles, so both partners can easily edit a file simultaneously. Like Sangam, DocShare easily interfaces with Rational Team Concert and preserves students' privacy. During a lab session two weeks after we introduced Sangam, we gave students instructions on how to install DocShare. The teaching staff again encouraged students to use a distributed pair programming tool when they were unable to meet in the same location.

[12] NCSU IRB approval 347-08-9

All students in the course were required to submit a retrospective after they submitted the team assignment. Forty-six of the 49 students submitted retrospectives. In their retrospectives, students reflected on their experiences in the course, including pair programming. Retrospective questions relevant to this study are:

R1: ***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.]?*

R2: ***Distributed pair programming.*** *If you chose not to do the distributed pair programming extra credit, briefly describe why not.*

To encourage students to try a distributed pair programming tool, students had an extra credit pair programming option for their team project. The option required students to pair program while collocated or distributed, keep a log of their pair programming, and answer nine questions about their experiences. Students could complete other extra credit options so they did not have to report pair programming but could still earn extra credit. Eight of the 49 total students chose to complete the pair programming extra credit option. Two of those eight students reported distributed pair programming while the other six reported collocated pair programming. Twenty-nine of the students who submitted retrospectives indicated that they pair programmed at some point during the team project, but only eight completed the extra credit option. The two students who distributed pair programmed used DocShare as their tool. The pair programming extra credit questions are the following:

EC1: *Which plugin did you use for distributed pair programming?*

EC2: *How often and for how long did you typically do distributed pair programming?*

EC3: *How often and for how long did you typically do collocated pair programming?*

EC4: *Do you think you were more likely to pair program because you were able to do it in a distributed way?*

EC5: *Did you enjoy collocated or distributed pair programming more?*

EC6: *Do you feel that pair programming had a positive or negative effect on how much you accomplished? On the quality of your code? Why?*

EC7: *Did you tend to pair program with the same person or did you rotate?*

EC8: *Did you run into any technical issues that prevented you from doing distributed pair programming?*

EC9: *Are there any features you would like to see improved or added to either plugin?*

Answers to the pair programming extra credit and retrospectives were quantitatively and qualitatively analyzed as described in section 2.2.

5.2 Retrospective Results

This section presents the results from the analysis of students' retrospectives.

5.2.1 Motivations for Pair Programming

In the retrospectives, students were to answer the free-form question, "What made you decide to work in pairs?" Students could list more than one reason, and 16 of the 46 students listed at least one reason. Below we present a ranked summary of the answers that

more than one student listed. The number of students who listed each answer is in parentheses.

- Working on a complex requirement or code (7)
- Debugging (6)
- Ensuring assignment was completed on time—"pair pressure" (4)
- Unsure of how to proceed (3)
- Discussing implementation direction (2)
- Desiring a peer review of code (2)
- Integrating different portions of the code (2)
- Upon discovering that two tasks were interdependent (2)

The two most common reasons students chose to work in pairs was when the requirements or coding required was complex and when debugging. Distributed pair programming is an option when students are working alone in separate locations and are having difficulty debugging a problem. If their partners are online, they can share code and work together to debug the problem. They do not have to wait for the next time they see their partner or struggle alone.

Students reported "pair pressure" and used it to their advantage. One student purposely chose to pair program because,

> *By having someone depending on me to pull things together, I find myself planning better and getting things done earlier than I would if I was on my own.*

5.2.2 Motivations for Not Pair Programming

In their retrospectives, 26 of the students explained their motivations for not pair programming in general. Below we present a ranked summary of the motivations that more than one student listed. The number of students who listed each motivation is in parentheses.

- Scheduling conflicts (21)
- Did not feel pair programming was time effective (6)
- Not collocated with his or her partner at similar times (4)
- Preference to work alone (2)

Eighty percent of the students who listed a reason for not pair programming stated that schedule conflicts were a reason why. Distributed pair programming can help in that programmers do not have to worry about travel time to a central meeting location; programmers can work together from wherever they are. However, if one partner can only program in the afternoon and the other partner can only program in the evening, distributed pair programming will not be able to assist.

5.2.3 Motivations for Pair Programming Remotely

Persons with visual impairments may prefer distributed pair programming, which allows them to pair program from their personalized workstations. Although his team decided not to do the extra credit option, one student with a visual impairment commented,

> *I enjoy working with my own machine and on a large monitor with my preferred resolution (I have a visual impairment so this is very important to me) and meeting up and working on a laptop with another person can be a bit of a struggle visually.*

Because of these elements, distributed pair programming would have been a great technology to try out…

5.2.4 Motivations for Not Pair Programming Remotely

Students who chose not to do the distributed pair programming extra credit were asked why they chose not to do so. Students could list more than one reason, and 29 of the 46 students listed at least one reason. Below we present a ranked summary of the motivations that more than one student listed. The number of students who listed each motivation is in parentheses.

- Did not want to learn the distributed pair programming technology or had difficulty getting the technology to work (11)
- Scheduling conflicts (8)
- Preference for collocated work (8)
- Forgot the distributed pair programming extra credit was an option (3)
- Preference to work alone (3)
- Did not find pair programming beneficial (2)
- Team was effective without distributed pair programming (2)
- Divided the project into independent tasks (2)

Eleven students (24% of the students who submitted retrospectives) said they did not do the distributed pair programming extra because they did not want to learn the distributed pair programming technology or had difficulty getting the technology to work. Some students had an unpleasant experience with Sangam and did not want to put in the effort to

learn another distributed pair programming tool, DocShare. Some said they ran into technical difficulties and chose to give up on distributed pair programming. This particular semester, the code repository server had technical problems, and students became frustrated with the course tools. We feel that these unrelated technical issues contributed to students' desire to not learn another distributed pair programming tool. We now have a tutorial to help students learn DocShare and are not asking students to try Sangam.

Two other common reasons why students chose not to try distributed pair programming were scheduling conflicts and a preference for collocated work. They found the benefits of being able to meet in person outweighed working the benefits of working remotely. Many students in the class were able to physically meet together in the LCSD. A majority of these students live on or near campus and visit campus for classes on a regular basis. Since students had access to the lab, many of them chose to have their team meetings and programming sessions in a collocated environment as they felt the face-to-face communication was much more beneficial. The following are excerpts from student retrospectives on their preference for collocated work:

> *While useful in theory, especially if your group is working from many different locations, we found it much easier to just meet often to work.*

> *Our team did not attempt distributed pair programming. We felt that it was more important to have both people present and accounted for, rather than letting each other be paying less attention to programming and more to the environment around us.*

We decided not to do distributed pair programming because we as a team felt that if we were going to work as pairs, we might as well do it face to face so it [would] be more personal. It [is] easier to talk to someone rather than switch to some message program that is slow and takes time away from you looking at the coding task at hand.

I like meeting people face to face when I have to interact with them for an extended period of time.

We did not use the distributed pair programming extra credit because we were able to work together. I am sure that distributed pair programming would have been more helpful than working solo, but we felt it was even more beneficial to work together in person. Had there been reasons we could not meet up in person, I am sure we would have given a second thought about distributed pair programming.

I didn't try the distributed pair programming utility, and while I can't say for sure (because I didn't try it!), I would imagine it would just be more cumbersome than regular pair programming. Trying to program and communicate with text chat, it would just be too slow I imagine to convey ideas to make pair programming effective. Even with voice chat, you still lose the ability to look at the same screen and point to something. Did you notice during labs when people work together, you would rarely see them looking at their separate but identical monitors? People would lean over and look at the same screen. I think the visual clues of where someone is looking, as well as

the ability to point to the screen, really helps. This is all lost with dual monitors and especially over remote connection. I can't imagine it working too well.

Working in pairs was effective because we could point to things on the screen or reason things out in person. We did not feel that it would [have] been beneficial for 2 people to work on the same piece of code without the human-interaction.

Three students (about 6.5% of the students who submitted retrospectives) reported a general preference for working alone as they enjoy solving the problems by themselves. This is consistent with prior work [31] that about 5% of students typically enjoy working alone more than pair programming. One student said she was uncomfortable with the idea of someone watching her program. In her case, this seems to be more related to a low self-efficacy of her programming skills than a preference to be independent. She said,

The thought of someone else being able to type where I am typing and "looking over my shoulder" makes me very uncomfortable.

5.3 Pair Programming Extra Credit Option Results

This section presents the results from the analysis of students' pair programming extra credit option answers.

Six students reported collocated pair programming and two students reported both collocated and distributed pair programming for the extra credit option. Two of the six students who pair programmed while collocated and one of the two students who pair programmed while distributed were female. One of the six students who pair programmed while collocated was an African American male.

All six of the eight students who responded to the question, "Do you feel that pair programming had a positive or negative effect on…the quality of your code?" answered "positive." Three of the eight students mentioned scheduling problems when writing about their experiences with pair programming although the questions they were answering did not lead to such an answer.

Table 2 presents a summary of student responses to the pair programming extra credit option.

Table 2. Summary of pair programming extra credit responses.

Student ID	Did you pair program while collocated or distributed?	Did you enjoy collocated or distributed pair programming more?	Did pair programming have a positive or negative effect on how much you accomplished?	Did pair programming have a positive or negative effect on the quality of your code?	Did you pair program with the same person or rotate?
A	Collocated	Collocated	No response	Positive	Rotate
B	Collocated	Collocated	No response	No response	Rotate
C	Collocated	Collocated	Negative	Positive	Rotate
D	Collocated	Collocated	No response	No response	No response
E	Collocated	Collocated	Positive	Positive	Rotate
F	Collocated	Collocated	Positive	Positive	Rotate
G	Both	Distributed	Negative	Positive	Same
H	Both	Distributed	Negative	Positive	Same

The two students who pair programmed remotely described their experience. One partner was a female and the other partner was a male with two young children. They noted that distributed pair programming allows programmers to work from the comfort of their own homes. They also commented that distributed pair programming allows students with

personal responsibilities that do now allow them to meet on campus a method to collaborate with their teammates. The female partner wrote,

To be honest, the only reason we did the distributed paired programming was to get the five extra credit points. However, after I did the distributed paired programming, I learned that it is something for teams to look into. … Distributed paired programming is nice because it allows two people, who may need to be in certain places at a certain time, still work together to code. This makes it a more preferred plug in for me and other people that have a hard time meeting up in person…For some situations we found that distributed paired programming is a better tool then co-located paired programming. This is because [my partner] has two kids and other off-campus responsibilities so it is hard for him to come to school to do co-located pair programming. It is nice being able to work from home while still working together…

The male partner wrote,

Though [my partner] might disagree with me on this point, I felt like distributed pair programming was actually rather enjoyable. It gave us the flexibility and comfort of working at home while also giving us the benefits of working in a lab with a partner. To maintain the mechanics of pair programming, we spent the majority of our distributed time working in a driver/watcher mode but there were a few instances where we created our own practice of "concurrent-solo programming." It was actually rather entertaining when both of us were typing in the same file at the same time but in different methods. This was something that simply wouldn't be possible with regular pair programming and really increased the fun-factor of the assignment.

The pair offered their chat log from when they distributed pair programmed so we could learn about their experience. During the session, they discussed a diagram they had previously created, which seemed to work well for them, but only one partner had a copy of the diagram. From reading their log, it was evident they were comfortable working together. They valued each other's opinions and asked about decisions as they went. They joked occasionally during the session and briefly discussed an email about another class. They seemed to be interacting much as they would in person. They also took short breaks much as collocated partners do. They took bathroom breaks and phone breaks but were able to jump back into programming, which may be because when one person took a break, the other person remained focused on the code.

5.4 Study Limitations

Not all students answered all questions. Three students did not submit retrospectives.

All study results are subjective and based on student perceptions rather than actual measures of quality or productivity.

Twenty-nine students indicated they pair programmed during the team project, but only eight students submitted the extra credit option. Students may have thought they could only submit the extra credit option if they pair programmed for a majority of the team project or used the distributed pair programming tools.

The sample size is small and the study took place in one course at one university with a pair programming lab. These results may not generalize to students at other universities.

5.5 Observations and Discussion

Several students noted or implied that they partner programmed; they worked together in the same room but were not pairing to work on the same software artifact. They noted that working in the same room as their teammates was beneficial so if they were ever stumped on what to do next, someone was available to help. Distributed pair programming tools can also provide a benefit similar to partner programming. If students are working from separate locations, they can share the artifact on which they need assistance, debug the issue, and then return to their individual work.

We observed several problems that students encountered when trying to install and use the distributed pair programming tools, Sangam and DocShare. Students had difficulty getting the plugins installed on their personal computers. Sangam only transfers edits that the driver makes, but it does not restrict the navigator from making edits. Therefore, students had synchronization errors if they accidentally began typing when they were still in the "Navigator" mode. Also, since Sangam edits local copies of files on both partners' computers, students found reconciling both partners' copies with the version control system to be difficult. When using Sangam, partners using different operating systems had to convert the line delimiters to Unix on each file they wanted to program together, which is an inconvenience. Some students complained that they had to connect to an outside chat client to use DocShare; they did not want to create accounts if they did not already have them. Although students noted to the teaching staff that they could see how a tool like this would be beneficial, many students did not have a positive attitude toward the tool.

We found it interesting that although we offered the extra credit options to students to pair program remotely, many of the students chose not to pursue the option. In informal discussions with students, we found that many students perceived pair programming to take longer than the "divide and conquer" approach. In talking with one student that chose to do the distributed pair programming, the extra credit was certainly a motivating factor. However, he said that one major benefit is that since he has two young children and cannot come in to the lab to work with the group, he appreciated being able to collaborate with the group but remain at home. We found that many students chose not to participate in distributed pair programming simply because they did not have a need to. Students attend the software engineering class and lab three times a week, so if they want to meet in person, they can do so without much difficulty. Therefore, distributed pair programming may not be a necessity for them. However, teams that are physically distributed on a regular basis could find distributed pair programming to be a major benefit. Our findings align with Olson and Olson's results. They suggest that when new groupware is introduced, incentives to use the groupware are necessary [20]. Many students found the extra credit incentive to not be worth the effort of learning new technology.

Scheduling conflicts were a major motivation for students choosing not to pair program remotely. We do not know if these scheduling issues were due to students not being able to meet in the same location at a given time or students not having times that align. Distributed pair programming could help solve the problem of students not being in the same location, but it cannot help the problem of students not having times that align.

Based on the Myers-Briggs results students reported in PairEval, the software engineering course had 61% introverts and 39% extraverts. This increased percentage of introverts may have contributed to fewer students trying distributed pair programming.

5.6 Conclusions

In this study, three research hypotheses and one research question were examined.

H1a: *Students who pair program remotely will enjoy many of the same benefits as those who pair program while collocated, including higher code quality and personal enjoyment*. Supported. The two students who reported distributed pair programming for the extra credit option reported that pair programming had a positive effect on their code quality. They also reported enjoying distributed pair programming.

H1b: *Women and students from other underrepresented groups in computing who pair program remotely will enjoy many of the same benefits as those who pair program while collocated, including personal enjoyment*. Supported. One of the two students who tried distributed pair programming was a female and she enjoyed it, so this hypothesis is supported.

H3: *Making distributed pair programming technology available to students increases the likelihood that they will pair program.* Partially supported. The availability of the distributed pair programming tools increased the amount that two students pair programmed. The other students in the class did not take advantage of pair programming remotely, so this hypothesis is partially supported.

RQ1: *What motivates students to or not to pair program remotely?* Persons with visual impairments may prefer distributed pair programming, which allows them to pair

program from their personalized workstations. Students are also motivated to pair program remotely because of the convenience and comfort of working from home or personal responsibilities that require them to work from home. The main motivation students described for not pair programming remotely was a lack of desire to learn the distributed pair programming technology or difficulty getting the technology to work. Students also listed scheduling conflicts and a preference for collocated work as primary reasons for not pair programming remotely.

6 Study 2: Interviewing Students about Their Pair Programming Experiences

The goal of our second study is to examine student perceptions about distributed pair programming when a distributed pair programming tool is available for use.

To supplement the student responses from our first study, we interviewed students in the same software engineering course to discuss their perceptions about computer science as a field, teamwork, and pair programming. We conducted interviews with six students at the beginning of the semester and six students at the end of the semester. The interviews were transcribed and then quantitatively and qualitatively analyzed. Through the interviews, four research hypotheses and one research question were examined:

H1a: *Students who pair program remotely will enjoy many of the same benefits as those who pair program while collocated, including higher code quality and personal enjoyment.*

H1b: *Women and students from other underrepresented groups in computing who pair program remotely will enjoy many of the same benefits as those who pair program while collocated, including personal enjoyment.*

H2: *Distributed pair programming decreases the scheduling issues that arise for students trying to schedule collocated pair programming.*

H3: *Making distributed pair programming technology available to students increases the likelihood that they will pair program.*

RQ1: *What motivates students to or not to pair program remotely?*

6.1 Research Methodology

We conducted interviews[13] with students in the software engineering course to discuss their perceptions about computer science as a field, teamwork, and pair programming. We visually identified nine students who were women and/or part of traditionally underrepresented groups in computing and randomly-selected nine other students in the class to invite for an interview at the beginning of the semester. At the end of the semester, we invited those same eighteen students to interview. We also invited the ten students who indicated to the teaching staff that they were working toward the pair programming extra credit described in Study 1. Four of these ten students were part of the original eighteen invited at the beginning of the semester. Six students participated in the beginning interview, and six students participated in the end interview. None of the students from traditionally underrepresented groups participated in the beginning or ending interviews.

Table 3 shows the gender breakdown of the interviewees. As an incentive, interviewees received a $20 gift certificate to a location of their choice for each interview in which they participated.

Table 3. Gender breakdown of interviewees.

ID	Gender	Beginning Interview	Ending Interview
20	Male	X	X
21	Female	X	
22	Male	X	X
23	Female	X	X
24	Male	X	X
25	Female	X	X
26	Male		X

[13] NCSU IRB approval 347-08-9

The interviewer followed the protocols available in Appendices A and B; however, the interviewer inadvertently did not ask each interviewee every question. The interviews were sound recorded. The interviews were transcribed and then quantitatively and qualitatively analyzed as described in section 2.2.

6.2 Results

In this section, we discuss the results from our interviews.

6.2.1 On Teamwork

We asked interviewees what they liked and disliked about computer science and the software engineering course. Six interviewees mentioned problem solving as a reason why they liked computer science. Interviewee 20 mentioned that he enjoys computer science because he likes working alone, but he also finds working alone to be a drawback,

> *Being able to sit alone with my music in the zone is the plus side but on the other hand it's kind of the minus side too because I also really like interacting with people and I like being able to talk to others and collaborate...*

Four interviewees described working with others as something they liked about the course.

Of the five interviewees who were asked whether they prefer to work in groups or alone, two (20 and 22) prefer to work alone and three (23, 24, and 25) prefer to work in groups. Interviewee 20, as mentioned above, favors working alone:

> *Personally I'd just prefer to work by myself. If I was going to work at Cisco or something and then they told me I would be working with another person all the time*

there's no way I would take that job. There's no way. I want to work on my own. That's why I'm a computer scientist.

Interviewee 24, on the other hand, enjoys working with others:

I think our major needs to focus a lot more on actual communication and socialization amongst each other. Because when you get in industry, from what I've experienced, it's definitely you are now working with these people, and it is your job regardless of how well you can code, you must code with these people and you have to get along with them…Some people you know, when you start coding you shut yourself off from the world and you put your headphones on and some people – I sit there with my best friend and we can sit there all day and write PHP and I can talk to him like hey how are you doing and what not. We can laugh and whatnot…I pump it out at the same time and I'm done two hours earlier. It's the environment that you work in that's very much benefited when you're working in a team.

Interviewee 25 enjoys group work because she finds that she and her partner are able to take advantage of each other's strengths:

I prefer working in groups because my previous partner and I when we were working we were both able to pull each other up. He would do the research and his thinking process and my thinking process are different. I do research in one direction and he goes to do research in another direction. He's like "I found the answer to this," and I was like, "Really? Show me." So he showed me how he got the answer, and I showed him how I got the answer for another part, and we were able to get it working together.

6.2.2 On Pair Programming

Interviewees described the benefits of pair programming including partners helping generate new innovations and ideas, higher code quality, positive pair pressure, and partner assistance when they are not sure how to proceed.

The five interviewees (21, 22, 23, 24, and 26) who were asked if they had pair programmed in previous classes reported that they had not. The two interviewees (24 and 26) who were asked if they had pair programmed at work reported that they had not.

Interviewees 25 and 26 reported pair programming during the software engineering course's team project. Why did they choose to pair program? Interviewee 26 pair programmed when he and his teammate were working on the same use case. Interviewee 25 chose to pair program because,

It actually cut down on the time we were actually working.

Both interviewees 25 and 26 reported pair programming in the same location rather than from separate locations. Interviewee 25 chose to pair program in the same location as her partner due to technical problems with the distributed pair programming technology:

I think people complained that it was too difficult to deal with. We didn't want to waste time struggling over errors that came with the plugin at the same time as struggling with the errors of the program itself.

Interviewee 26 thought collocated work was easier:

There are some plugins to do the paired program. We could have used those but we decided to just actually meet since it would be easier to talk and see what people were doing rather than having to use a phone or IM.

The other four interviewees who chose not to pair program were asked why. Interviewee 20 disliked the scheduling aspect of pair programming:

> *I have kind of a bad attitude about paired programming in general just because everybody is so busy. Just to arrange our schedules and meet in the lab and just the pains we've had to go through.*

In fact, three other interviewees mentioned scheduling issues in their interviews as a problem they encountered. Interviewee 23 felt that the students in the software engineering course had become so accustomed to working alone due to the environment of other programming courses that pair programming was too much of a transition. She described,

> *I think a large part of that is how CSC students have come up through N.C. State. It's been a very individual thing. It's been very much this is your project, do not talk to anyone else or we'll shoot you in the face or whatever the rules are now. While I think it's a very good idea, … it's not one that's going to work with any kind of success in the environment that State has…*

Interviewees 20, 23, 24, and 26 all noted that they had not seen pair programming while working in industry, which seemed to make them question the value of pair programming. Interviewee 24 stated,

> *I think there are certain situations where paired programming is very appropriate and I think the issue is a lot of the jobs that most of the people coming out of here are looking for aren't supported by paired programming. I listen to people in class and they say well you know I worked at IBM, I worked at Cisco and we didn't do any of the paired programming. I worked in IT you know, people who had internships at Google*

and Microsoft; we didn't do paired programming there. So I do know there are certain shops they get away and do very well with paired programming.

Interviewee 24 went on to say,

Paired programming does exactly what they're saying it does. [My partner has] caught a lot of times where I've left off a semi colon or I've misspelled a variable. We do get higher quality code.

6.2.3 On Distributed Pair Programming

Five of the interviewees (21, 22, 23, 24, and 25) during their beginning interview were asked if they would pair program more often if they could pair program remotely. All but one (23) said they would. When asked if she would pair program remotely, interviewee 22 replied,

Absolutely. I usually find my schedule to be quite hectic.

Interviewee 24 thought pair programming in a distributed way was almost a necessity for students:

So it's almost like a requirement that you have some sort of way of remotely working. … I can see when you're on your job site doing paired programming because you're at the job because those are eight to five and you work these hours… So being able to work remotely would be amazing.

Interviewee 23 said she would not pair program remotely because she had tried the distributed pair programming tool Sangam and found it had technical issues. Interviewee 25 noted that although he would pair program remotely,

I just prefer working with somebody else in the same room instead of doing it remotely.

6.2.4 On the Increase of Skill Sets and Number of Friends

Based on the work of Walton and Cohen [25], we were curious how pair programming and this software engineering course affected students' computer science skill sets and number of friends within the computer science department. All of the interviewees who participated in the end interviews were asked if they felt they had more friends in the computer science department after taking this software engineering course. All but one interviewee (22) reported having more friends in the department. We also asked all of the interviewees who participated in the end interviews if they felt the course had increased their computer science skill sets and all but one interviewee (23) reported an increased skill set.

6.3 Study Limitations

Not all interviewees were asked exactly the same questions.

Not all interviewees participated in both the beginning and ending interviews.

All study results are subjective and based on student perceptions rather than actual measures of quality or productivity.

Due to time constraints, not all students in the course could be interviewed.

The sample size is small and the study took place in one course at one university with a pair programming lab. These results may not generalize to students at other universities.

6.4 Conclusions

In this study, four research hypotheses and one question were examined.

H1a: *Students who pair program remotely will enjoy many of the same benefits as those who pair program while collocated, including higher code quality and personal*

enjoyment. Not supported. None of our interviewees reported pair programming remotely during the course's team project, so we have no evidence for or against this hypothesis.

H1b: *Women and students from other underrepresented groups in computing who pair program remotely will enjoy many of the same benefits as those who pair program while collocated, including personal enjoyment*. Not supported. None of our interviewees reported pair programming remotely during the course's team project, so we have no evidence for or against this hypothesis.

H2: *Distributed pair programming decreases the scheduling issues that arise for students trying to schedule collocated pair programming.* Supported. Two of the four students who said they would pair program more often if they could pair program remotely said that remote pair programming would decrease scheduling issues.

H3: *Making distributed pair programming technology available to students increases the likelihood that they will pair program.* Partially supported. Four out of the five students who were asked if they would pair program more often if they could do so remotely said that they would. However, all of our interviewees had tools to pair program remotely, but none of them chose to do so.

RQ1: *What motivates students to or not to pair program remotely?* Interviewees reported scheduling issues as a motivation to pair program remotely. They listed technical problems with the distributed pair programming tools and a preference for collocated work as their motivation for not pair programming remotely. Interviewees described not seeing pair programming in industry and the department's culture of students working alone as reasons why they were not motivated to pair program in general.

7 Study 3: Student Perceptions Before and After Trying Distributed Pair Programming

The goal of our third study is to examine student perceptions about the benefits and drawbacks of distributed pair programming before and after trying distributed pair programming.

We conducted a study with 24 undergraduate software engineering students at NCSU during the spring semester of 2009. Participants pair programmed remotely for 90 minutes as they worked on a programming assignment together. We surveyed participants before and after these distributed pair programming sessions.

Through their survey responses, four research hypotheses and one question were examined:

H1a: *Students who pair program remotely will enjoy many of the same benefits as those who pair program while collocated, including higher code quality and personal enjoyment.*

H1b: *Women and students from other underrepresented groups in computing who pair program remotely will enjoy many of the same benefits as those who pair program while collocated, including personal enjoyment.*

H2: *Distributed pair programming decreases the scheduling issues that arise for students trying to schedule collocated pair programming.*

H4: *Trying distributed pair programming increases the likelihood that students will pair program remotely in the future.*

RQ2: *What do students perceive as the benefits and drawbacks of distributed pair programming?*

7.1 Research Methodology

We conducted a study[14] with 24 undergraduate software engineering students at NCSU. We limited this study to students in the software engineering course as we wanted participants to have a vested interested in accomplishing a similar programming task. We solicited all 69 students in the course, and 12 pairs chose to participate. No student volunteered to participate without his or her partner. Participants were given a $20 gift card as an incentive. Two of the participants were female. One male participant was African American/Black, which we consider to be a minority for this study. One male participant declined to report his ethnicity.

For the study, we asked participants to take a Pre-Study Survey (Appendix C), try distributed pair programming for approximately 90 minutes with their assigned partner for their second paired homework assignment, take a Post-Study Survey (Appendix D), and answer questions in a follow-up email (Appendix E). One of the students met a researcher in the LCSD while his partner remained in another location of his choice. We suggested participants share their code through DocShare, voice chat using TokBox[15], and text chat using the DocShare chat tool, but we allowed them to communicate using any of the tools they had available. The researcher observed the participant in the LCSD during the study.

[14] NCSU IRB approval 48-09-01
[15] http://www.tokbox.com

The homework assignment participants worked on during the study was the second paired assignments that students have three weeks to complete. At the end of the first week, students submit a UML class diagram and test plan for a provided requirement. At the end of the third week, students submit their project, which requires automating the insertion and deletion of test data, implementing the assignment requirement, and creating automated tests for their implementation. We conducted the study after week one of the assignment.

We recommended participants use DocShare because it easily interfaces with Rational Team Concert, it preserves students' privacy (unlike desktop sharing tools), and students in the previous semester preferred DocShare to Sangam. Prior to the study, all students in the course were asked to complete a tutorial[16] on how to install and use DocShare. One pair chose to use LogMeIn instead of DocShare during the study.

We quantitatively and qualitatively analyzed the participants' responses to the Pre-Study Survey, Post-Study Survey, and follow-up email as described in section 2.2.

7.2 Results

This section provides the results from our study.

7.2.1 Personal Enjoyment

Sixty-seven percent of participants agreed that they enjoyed distributed pair programming. Only two participants disagreed (zero participants strongly disagreed) with the statement, "I enjoyed pair programming remotely with my partner" (Q12). We compared participants' enjoyment of programming alone (Q25), collocated pair programming (Q26),

[16] http://agile.csc.ncsu.edu/SEMaterials/tutorials/docshare/docshare.htm

and distributed pair programming (Q12) of the 22 participants who responded to the follow-up email. Figure 2 illustrates that more participants enjoy collocated pair programming and distributed pair programming compared to programming alone.

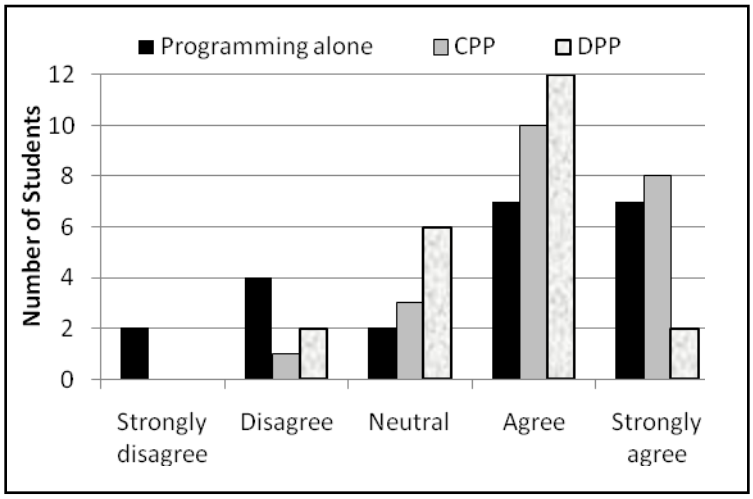

Figure 2. Preferences for programming alone, collocated pair programming (CPP), and distributed pair programming (DPP).

We also examined the personal enjoyment of the three participants who were women or part of other underrepresented groups in computing. Table 4 shows the responses of these three students to their enjoyment of programming alone, collocated pair programming, and distributed pair programming.

Table 4. Preferences for programming alone, collocated pair programming, and distributed pair programming of women and underrepresented minorities.

Gender	Ethnicity	Enjoy Programming Alone	Enjoy Collocated Pair Programming	Enjoy Distributed Pair Programming
Female	Caucasian	Strongly agree	Agree	Agree
Female	Asian/Pacific Islander	Strongly disagree	Strongly agree	Neutral
Male	African American/Black	Neutral	Strongly agree	Neutral

Participants who found DocShare easy to use were more likely to enjoy distributed pair programming. We used simple linear regression and found a statistically significant ($p<.01$, $r=.5375$) positive correlation between the perceived ease of use of the distributed pair programming tool and participants' enjoyment of distributed pair programming. Therefore, students may need to perceive the distributed pair programming tool as easy to use in order to enjoy distributed pair programming.

7.2.2 Perceived Code Quality

More participants agreed than disagreed that distributed pair programming code quality was higher than that of programming alone (Q14). However, more participants disagreed than agreed that distributed pair programming code quality was higher than that of collocated pair programming (Q15). Almost half of the participants had neutral responses for Q14 and Q15. Figure 2 shows the detailed results of these two questions.

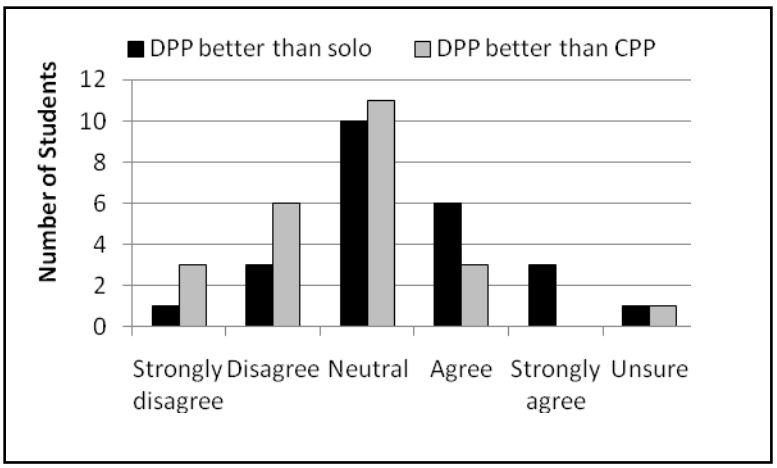

Figure 3. Code quality comparisons of distributed pair programming (DPP) to solo programming and collocated pair programming (CPP).

7.2.3 Productivity

Forty-three percent of participants agreed or strongly agreed that they produced the same or more code during distributed pair programming compared to working alone (Q16b), and 50% of participants agreed or strongly agreed they produced the same or more code during distributed pair programming compared to collocated pair programming (Q17b). We modified Q16a and Q17a to say "same or more" rather than "same" after 10 participants had taken the survey to gather more accurate results. The fourteen survey results after the question modification are presented in Figure 4.

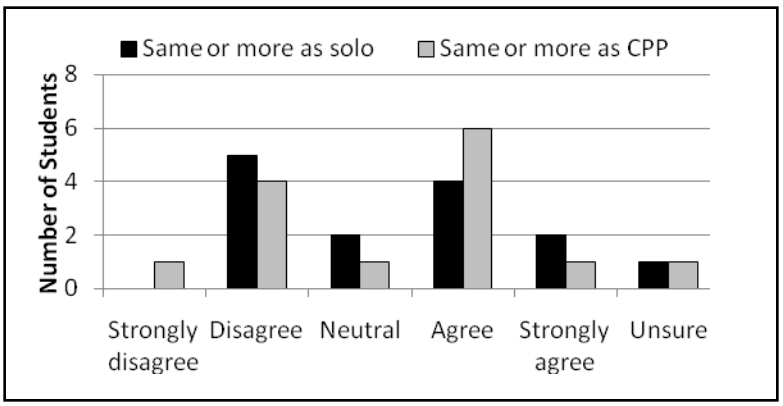

Figure 4. Productivity comparison of distributed pair programming to solo programming and collocated pair programming (CPP).

7.2.4 Pair Pressure

Seventy percent of participants agreed or strongly agreed that their partner helped keep them on task (Q18). Six participants were neutral, and only one participant disagreed with Q18.

7.2.5 Continuing Pair Programming

The percentage of participants who were likely or extremely likely to pair program remotely increased from 54% to 66% after trying distributed pair programming (Q6 and Q11). The percentage of participants who were likely or extremely likely to pair program while collocated increased from 58% to 83% after trying distributed pair programming (Q5 and Q10).

We sent a follow-up email to participants the day after the assignment deadline and asked if they pair programmed remotely again on that assignment (Q23). Of the 22 participants who responded, only two participants said they pair programmed remotely although both of their partners reported not pair programming remotely. We asked

participants why they had or not pair programmed remotely again. Participant answers were open ended and could contain multiple reasons. The four most common reasons given for not pair programming remotely again after the study were (1) a preference for collocated work (seven participants); (2) no scheduling issues (six participants); (3) working separately on the assignment (four participants); (4) and being almost done with the assignment (two participants).

We also asked participants if distributed pair programming helped simplify scheduling issues (Q24), and eight participants responded "yes," 11 participants responded "no," two participants responded "N/A", and one participant did not answer the question.

Scheduling conflicts were not a major issue for our participants. One participant noted,

We had already found several times that worked for both of us together. If we hadn't been able to find time, then remote pair programming probably would have helped with scheduling issues.

Another participant commented,

I think I will [pair program remotely] if my future partner cannot meet with me at the lab and work on it together. It is by far the best alternative way to communicate [when I have schedule] conflicts with [my] partner.

One of the major motivations for working remotely is being unable to meet in a common location, but our participants have access to the LCSD on campus. One participant said he did not pair program remotely "because the lab allows for the only one better alternative, working together in person." We asked participants how long it takes them to travel to campus (Q4) and 23 participants answered. Twenty participants say it takes them

between zero and 20 minutes to travel to campus and the other three participants can travel to campus in 30 minutes. Travel time is not a large burden on participants so they may be less inclined to continue distributed pair programming.

7.2.6 Perceived Benefits of Distributed Pair Programming

We asked participants to list the benefits of distributed pair programming in the Post-Study Survey (Q19). Participant answers were open ended and could contain multiple reasons. Below we summarize their responses. The number of participants who listed each benefit is in parentheses.

- **Convenience (5)**. Distributed pair programming can be more convenient than collocated pair programming. Participants were not specific about what convenience meant to them, but it could mean ease of scheduling or eliminated travel time.
- **Eliminated travel time (3)**. Partners do not have to travel to meet their partners when pair programming remotely.
- **Partner feedback (3)**. Much like collocated pair programming, partners collaborate together.
- **Concurrent editing (1)**. DocShare allows both partners to edit the code synchronously. For example, both partners could be writing methods in the same class. Collocated pair programming on the same machine would not allow for this concurrent editing.
- **Comfort of working from home (1)**. Distributed pair programming allows programmers to work in their own space that may be more comfortable than a lab.

- **Spontaneous pair programming (1)**. If a partner is working independently and needs help debugging a problem, he can easily begin pairing with his partner. The partner will not have to debug the problem on his own or wait until the next paired session.
- **Distributed communication can allow for more open communication (1)**. As one participant stated,

  I didn't feel as awkward asking my partner questions "over the phone" as I would've if we were in person. Facial expressions in person may indicate the partner is annoyed with me asking questions, but since I couldn't see him, I didn't feel as self conscious about asking questions.
- **Pair Pressure (1)**. Partners help each other stay on task.
- **Similar to working in person (1)**. Distributed pair programming provides many of the same benefits as working in person rather than working alone.

7.2.7 Perceived Drawbacks of Distributed Pair Programming

We asked participants to list the drawbacks of distributed pair programming in the Post-Study Survey (Q19). Participant answers were open ended and could contain multiple reasons. Below we summarize their responses. The number of participants who listed each drawback is in parentheses. Some of these drawbacks are specific to DocShare rather than distributed pair programming in general.

- **Inability to share multiple editors (6)**. DocShare does not allow for the sharing of multiple editors.

- **Initial setup can be difficult (6)**. Several participants had difficulty configuring their workspaces correctly with the latest code from their repositories. Some participants had difficulty setting up DocShare. Since the partners could not see each other's screens, they had difficulty helping debug these setup issues.
- **Difficult for partners to follow along with each other (5)**. Since DocShare does not indicate what portion of the code each partner is viewing or editing, partners had to explicitly describe what portion of the code they were viewing.
- **No gesture support (5)**. DocShare does not support gesturing, which made it difficult for some participants to fully communicate with their partners.
- **Files stored on only one partner's computer (2)**. DocShare stores the shared files on only one partner's computer, so participants had to check-in and accept code throughout the distributed pair programming session.
- **Easy to get off task (2)**. A partner may not notice as quickly if his partner gets off task.
- **Difficult to debug technical problems (1)**. Without full screen sharing, it can be difficult for partners to help debug technical problems.
- **Less productivity initially (1)**. Participants had an adjustment period to the tool and working remotely, which led to less productivity during the programming session. Participants noted that their productivity would have increased with experience.
- **Distributed pair programming technology can act in unexpected ways (1)**. One participant noted in his Post-Study Survey and two pairs reported to a researcher during the study that DocShare unexpectedly disconnected them during their distributed pair programming session.

7.2.8 Recommendations for Others Pair Programming Remotely

We asked participants in the Post-Study Survey to provide recommendations for others who try distributed pair programming (Q21). Participant answers were open ended and could contain multiple recommendations. Below we summarize their responses. The number of participants who listed each recommendation is in parentheses.

- Communicate with your partner frequently (3).
- Set up your workspaces and distributed pair programming technology with your partner in the same location (3).
- Use a remote desktop control system (2).
- Designate one partner to share files (2).
- Have a pre-set list of goals for the session (2).
- Use voice chat (1).
- Use a collaborative text editor (1).
- Set up your communication tool before beginning to share files (1).
- Keep trying distributed pair programming until it makes more sense (1).
- Try distributed pair programming for the first time with a partner who has experience with distributed pair programming (1).
- Be in a quiet location (1).
- Use a comfortable headset for voice chatting (1).

7.2.9 Suggestions for Improving Distributed Pair Programming Experience

In Q21, we asked students what would improve their distributed pair programming experience. Many of their suggestions regarded improving DocShare.

Seven students responded that they wanted a way to share more files with their partner through either multiple editor sharing, full project sharing, or by sharing their entire Rational Team Concert application. Four students desired a way to share their screens or desktops. By sharing more than just one editor, they would be able to flip back and forth between several files and possibly share their automated testing tools. One pair during the study chose to use LogMeIn, which one of the partners had previously installed on his computer, so they could share multiple files.

Several pairs spent a significant amount of time getting set up initially due to technical problems, and being able to see each other's screens may have been beneficial. Some of the teams had trouble setting up the work environment on the lab machines or on their laptops. In DocShare, the pair can only jointly see the code editor, not their full screen. When one partner had an error message appear or could not find the correct menu option, the other partner could not see the problem and had difficulty helping debug the issue. In one case, the pair became so frustrated that they gave up working remotely and decided to temporarily meet in person before going back to working remotely. Additionally, pairs were not only editing code, they were referencing the assignment web page, posting to the course message board, running automated tests, and viewing the web application they were building in a browser. One participant commented to his partner, "I wish you could see my screen," and then proceeded to read parts of the assignment web page aloud to his partner. These

difficulties suggest that full application sharing, rather than just editor sharing, would be beneficial.

Two students desired a built-in voice chat feature for DocShare, so they would not have to flip back and forth between applications.

Based on observations of the participants and the drawbacks of distributed pair programming that participants listed, we suggest adding an option to sync the editors so both partners would see exactly the same code. For example, in the current version of DocShare, when one partner scrolled down, the other partner's editor did not scroll. Students had to explicitly tell their partners when they were moving to a new portion of the code. One participant commented in his Post-Study survey, "Often times we would have to 'dance around' by making and deleting a character in the file to indicate where we were making changes."

Additionally, DocShare does not support gesturing, which forced students to explicitly state line numbers. The lack of gesturing seemed to decrease efficiency, so we feel a gesturing feature would be beneficial.

7.3 Study Limitations

We were not able to compare code quality between those who worked while collocated and those who worked remotely because we could not require participants to work remotely on the entire assignment.

An increase in perceived code quality does not always mean an increase in actual code quality.

Participants had the opportunity to use DocShare before beginning the study, so they may have already formed opinions about DocShare and/or distributed pair programming. All students in the software engineering course were to complete the DocShare tutorial during a weekly lab session, but many participants had not learned how to use DocShare prior to the study.

Participants may not be an accurate representation of the entire class. Students who chose not to participate in the study may have done so because they prefer not to pair program.

7.4 Observations and Discussion

Participants had an adjustment period to working remotely and often did not think of the most efficient way to work with their remote partners. For example, one pair during the study brainstormed test data together. One partner took notes in notepad, and consequently, the other partner was unable to see the notes. After a few minutes, the partner taking notes realized this was a problem, so she opened a text editor in Rational Team Concert, copied the notes into the text editor, and then shared it with her partner so they could both see and edit the notes.

Participants occasionally became distracted by other students during the study, demonstrating that it may be helpful to be in a quiet place when pair programming remotely. We held the study in the LCSD, so occasionally other students were in the lab working during the study. At one point, two students were working together and became very excited and noisy when they accomplished a task, and this disrupted our study participants. Additionally, as students came and entered the lab, they greeted the study participants just as

they would if they had not been participating in the study. These distractions are similar to what may occur if students try distributed pair programming on their own.

One participant noted that distributed pair programming may be more helpful for hobby programming rather than homework assignments:

> *Overall, I would be interested in continuing to use this, but maybe not so much for class projects with deadlines. I would probably practice using it for hobby-style projects.*

7.5 Conclusions

In this study, four research hypotheses and one question were examined.

H1a: *Students who pair program remotely will enjoy many of the same benefits as those who pair program while collocated, including higher code quality and personal enjoyment*. Partially supported. A higher percentage of students enjoy collocated pair programming and distributed pair programming compared to programming alone. We found a statistically significant positive correlation between ease of use of the distributed pair programming tool and enjoyment of distributed pair programming, so students may need to find the distributed pair programming tool easy to use to enjoy distributed pair programming. More participants agreed than disagreed that distributed pair programming code quality was higher than that of programming alone. However, more participants disagreed than agreed that distributed pair programming code quality was higher than that of collocated pair programming.

H1b: *Women and students from other underrepresented groups in computing who pair program remotely will enjoy many of the same benefits as those who pair program while*

collocated, including personal enjoyment. Not supported. Only two of the participants were females, and only one of the participants was an African American/Black male. All three responded differently on how much they enjoyed distributed pair programmed compared to how much they enjoyed programming alone, so we cannot draw a conclusion about this hypothesis.

H2: *Distributed pair programming decreases the scheduling issues that arise for students trying to schedule collocated pair programming.* Partially supported. Participants' responses were mixed and only two participants reported trying distributed pair programming again after the study.

H4: *Trying distributed pair programming increases the likelihood that students will pair program remotely in the future.* Partially supported. Although the percentage of students likely to pair program while collocated and pair program while distributed increased after students tried distributed pair programming during the study, the follow-up email results at the end of the assignment were contradictory. Several of the participants in our study had a preference for collocated work and were able to do so because they had minimal scheduling conflicts and access to a lab designed for collocated pair programming.

RQ2: *What do students perceive as the benefits and drawbacks of distributed pair programming?* Participants described the benefits of distributed pair programming: convenience; elimination of travel time; and partner feedback. Participants described the drawbacks of distributed pair programming: difficulty for partners to follow along with each other and less productivity initially due to the learning curve of the tools and learning how to work remotely.

8 Summary

To examine student perceptions about distributed pair programming, we conducted three studies, discussed in chapters 5, 6, and 7, with upper-level undergraduate software engineering students. We found that upper-level undergraduate students can successfully pair program remotely. However, students who can travel fairly quickly to a common lab space are not likely to choose to pair program remotely. In this chapter, a summary of the examination of the research hypotheses and questions is presented.

8.1 Summary of Research Hypotheses and Questions

H1a: *Students who pair program remotely will enjoy many of the same benefits as those who pair program while collocated, including higher code quality and personal enjoyment*. Partially supported. In Study 1, the two students who reported pair programming remotely for the extra credit option reported that remote pair programming had a positive effect on their code quality. They also reported enjoying distributed pair programming. None of our participants in Study 2 reported pair programming remotely during the course's team project, so we have no evidence for or against this hypothesis based on Study 2. In Study 3, a higher percentage of students enjoyed collocated pair programming and distributed pair programming compared to programming alone. We found a statistically significant positive correlation between ease of use of the distributed pair programming tool and enjoyment of distributed pair programming, so students may need to find the distributed pair programming tool easy to use to enjoy distributed pair programming. More participants agreed than disagreed that distributed pair programming code quality was higher than that of

programming alone. However, more participants disagreed than agreed that distributed pair programming code quality was higher than that of collocated pair programming.

H1b: *Women and students from other underrepresented groups in computing who pair program remotely will enjoy many of the same benefits as those who pair program while collocated, including personal enjoyment*. Supported. In Study 1, one of the two students who tried distributed pair programming was a female and she enjoyed it. None of our participants in Study 2 reported pair programming remotely during the course's team project, so we have no evidence for or against this hypothesis based on Study 2. In Study 3, two females and one African American/Black male participated. All three responded differently on how much they enjoyed distributed pair programming compared to how much they enjoyed programming alone, so we cannot draw a conclusion about this hypothesis.

H2: *Distributed pair programming decreases the scheduling issues that arise for students trying to schedule collocated pair programming.* Partially supported. In Study 2, two of the four students who said they would pair program more often if they could pair program remotely said that remote pair programming would decrease scheduling issues. In Study 3, participants' responses about scheduling issues were mixed, and only two participants reported trying distributed pair programming again after the study.

H3: *Making distributed pair programming technology available to students increases the likelihood that they will pair program.* Partially supported. In Study 1, the availability of the distributed pair programming tools increased the amount that two students pair programmed. The other students in the class did not take advantage of the ability to pair programming remotely. In Study 2, four out of the five interviewees who were asked if they

would pair program more often if they could pair program remotely said that they would. However, all of our interviewees had tools to pair program remotely, but none of them chose to do so during the team project.

H4: *Trying distributed pair programming increases the likelihood that students will pair program remotely in the future.* Partially supported. In Study 3, the percentage of students likely to pair program while collocated and while distributed increased after students tried distributed pair programming during the study, but the follow-up email results at the end of the assignment were contradictory. Several of the participants in the study had a preference for collocated work and were able to do so because they had minimal scheduling conflicts and access to a lab designed for collocated pair programming.

RQ1: *What motivates students to or not to pair program remotely?* Persons with visual impairments may prefer distributed pair programming, which allows them to pair program from their personalized workstations. Students are also motivated to pair program remotely because of the convenience and comfort of working from home, personal responsibilities that require them to work from home, and decreased scheduling issues.

The main reason students described for not pair programming remotely was technical difficulties with the distributed pair programming tools. Students also listed scheduling conflicts and a preference for collocated work as primary reasons for not pair programming remotely. Students described not seeing pair programming in industry and the department's culture of students working alone as reasons why they were not motivated to pair program in general.

RQ2: *What do students perceive as the benefits and drawbacks of distributed pair programming?* Students described the benefits of distributed pair programming: convenience, elimination of travel time, and partner feedback. Students described the drawbacks of distributed pair programming: difficulty for partners to follow along with each other and less productivity initially due to the learning curve of the tools and learning how to work remotely.

8.2 Concluding Remarks

We feel that distributed pair programming is a good alternative to working alone, but collocated pair programming still has advantages over distributed pair programming. Distributed pair programming requires good tool support and that partners communicate almost constantly. Distributed pair programmers must to be willing to learn the new technology and overcome initial technical difficulties.

Traditional age students in traditional (non distance-education) courses with sufficient lab facilities may not find distributed pair programming to be as beneficial as students who are unable to meet in the same physical location. As a result, traditional students may become easily discouraged with even seemingly small technology issues with the distributed pair programming tools and choose not to try again. Learning a new tool can require more effort of these students than traveling to campus to meet with their partners. Traditional students do cite scheduling issues as a reason why they do not pair program. However, these scheduling issues are often from not having the same times available to work; location is only a minor part of scheduling issues for these students. Most of the students in our study were traditional students. The two students in our first study who were willing to overcome

the technical issues of the distributed pair programming technology enjoyed distributed pair programming largely because one of the partners had young children and was unable to meet with his team in the lab. For many traditional students, the benefits of collocated work outweigh the benefits of distributed work.

Traditional students with visual impairments may be an exception. They may benefit from using distributed pair programming technology, which allows them to use their own personalized machines. For example, one student in our first study had a visual impairment and stated that the distributed pair programming technology would benefit him because he could use his own monitor with his preferred monitor resolution.

Both traditional and non-traditional students could reap the benefits of spontaneous distributed pair programming when partners are working separately but need help debugging. Students may only choose to try spontaneous distributed pair programming if they already have the distributed pair programming technology working.

All students, especially non-traditional students or students with visual impairments, in upper-level programming courses could find distributed pair programming beneficial as long as the teaching staff provides technical support for the distributed pair programming tool. To help alleviate the technical issues that may discourage students, we recommend giving students class time to learn how to install and use the distributed pair programming tool. We feel using a distributed pair programming tool without explicit driver and navigator roles that allows for gesturing and the sharing of more than one file would be beneficial.

REFERENCES

[1] "2006-2007 Taulbee Survey," Computing Research Association, www.cra.org.

[2] P. Baheti, L. Williams, E. Gehringer, and D. Stotts, "Exploring Pair Programming in Distributed Object-Oriented Team Projects," in *OOPSLA Educator's Symposium 2002* Seattle, WA, 2002.

[3] S. B. Berenson, K. M. Slaten, L. Williams, and C.-W. Ho, "Voices of women in a software engineering course: reflections on collaboration," *J. Educ. Resour. Comput.*, vol. 4, p. 3, 2004.

[4] J. Chong and T. Hurlbutt, "The Social Dynamics of Pair Programming," in *International Conference on Software Engineering (ICSE) 2007* Minneapolis, MN: IEEE Computer Society, 2007, pp. 354-363.

[5] A. Cockburn, *Agile Software Development*. Reading, Massachusetts: Addison Wesley Longman, 2001.

[6] A. Cockburn and L. Williams, "The costs and benefits of pair programming," in *Extreme programming examined*: Addison-Wesley Longman Publishing Co., Inc., 2001, pp. 223-243.

[7] N. V. Flor, "Globally distributed software development and pair programming," *Commun. ACM*, vol. 49, pp. 57-58, 2006.

[8] A. French and P. Layzell, "A study of communication and cooperation in distributed software project teams," in *International Conference on Software Maintenance, Proceedings*, Bethesda, MD, USA, 1998, pp. 146-154.

[9] S. Freudenberg, P. Romero, and B. du Boulay, "'Talking the talk': Is Intermediate-level conversation the key to the pair programming success story?," in *Agile 2007*, Washington, DC, 2007, pp. 84-91.

[10] B. Hanks, "Empirical evaluation of distributed pair programming," *Int. J. Hum.-Comput. Stud.*, vol. 66, pp. 530-544, 2008.

[11] B. F. Hanks, "Empirical studies of distributed pair programming," University of California at Santa Cruz, 2005, p. 168.

[12] C.-W. Ho, S. Raha, E. Gehringer, and L. Williams, "Sangam: a distributed pair programming plug-in for Eclipse," in *Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange* Vancouver, British Columbia, Canada: ACM, 2004.

[13] A. Höfer, "Video Analysis of Pair Programming," in *Workshop on Scrutinizing Agile Practices at the International Conference on Software Engineering*, Leipzig, Germany, 2008, pp. 37-41.

[14] D. Keirsey, *Please Understand Me II*. Del Mar, CA: Prometheus Nemesis Book Company, 1998.

[15] E. Mannix and M. A. Neale, "What Differences Make a Difference?," in *Psychological Science in the Public Interest*. vol. 6: Blackwell Publishing Limited, 2005, pp. 31-55.

[16] J. Margolis and A. Fisher, *Unlocking the Clubhouse: Women in Computing*: MIT Press, 2002.

[17] C. McDowell, L. Werner, H. E. Bullock, and J. Fernald, "The impact of pair programming on student performance, perception and persistence," in *Proceedings of the 25th International Conference on Software Engineering* Portland, Oregon: IEEE Computer Society, 2003.

[18] H. Natsu, J. Favela, A. L. Moran, D. Decouchant, and A. M. Martinez-Enriquez, "Distributed pair programming on the Web," in *Proceedings of the Fourth Mexican International Conference on Computer Science*, Colima, Mexico, 2003, pp. 81-88.

[19] K. Navoraphan, E. F. Gehringer, J. Culp, K. Gyllstrom, and D. Stotts, "Next-generation DPP with Sangam and Facetop," in *Proceedings of the 2006 OOPSLA workshop on eclipse technology eXchange* Portland, Oregon: ACM, 2006.

[20] G. M. Olson and J. S. Olson, "Distance Matters," *Human-Computer Interaction*, vol. 15, pp. 139-179, 2000.

[21] C. B. Seaman, "Qualitative Methods in Empirical Studies of Software Engineering," *IEEE Trans. Softw. Eng.*, vol. 25, pp. 557-572, 1999.

[22] D. Stotts, L. Williams, N. Nagappan, P. Baheti, D. Jen, and A. Jackson, "Virtual Teaming: Experiments and Experiences with Distributed Pair Programming," in *Third XP Agile Universe Conference*, New Orleans, LA, USA, 2003.

[23] J. Vanhanen and H. Korpi, "Experiences of Using Pair Programming in an Agile Project," in *40th Annual Hawaii International Conference on System Sciences (HICSS) 2007* Hawaii, 2007, pp. 274b - 274b.

[24] T. Waits, L. Lewis, and P. O. B. Greene, "Distance Education at Degree-Granting Postsecondary Institutions: 2000–2001," U.S. Department of Education, National Center for Education Statistics, Washington, DC NCES 2003-017, 2003.

[25] G. M. Walton, "A question of belonging: Race, Social Fit, and achievement," *Personality & social psychology bulletin,* vol. 92, p. 82, 2007.

[26] L. L. Werner, B. Hanks, and C. McDowell, "Pair-programming helps female computer science students," *J. Educ. Resour. Comput.,* vol. 4, p. 4, 2004.

[27] L. Williams and R. R. Kessler, *Pair Programming Illuminated*. Reading, Massachusetts: Addison-Wesley, 2003.

[28] L. Williams, R. R. Kessler, W. Cunningham, and R. Jeffries, "Strengthening the Case for Pair Programming," *IEEE Software,* vol. 17, pp. 19-25, 2000.

[29] L. Williams, L. Layman, J. Osborne, and N. Katira, "Examining the compatibility of student pair programmers," in *Agile Conference, 2006*, 2006, pp. 10 pp.-420.

[30] L. Williams, L. Layman, K. M. Slaten, S. B. Berenson, and C. Seaman, "On the Impact of a Collaborative Pedagogy on African American Millennial Students in Software Engineering," in *Proceedings of the 29th international conference on Software Engineering*: IEEE Computer Society, 2007.

[31] L. A. Williams, "The Collaborative Software Process," in *Department of Computer Science*. vol. PhD Salt Lake City, Utah: University of Utah, 2000.

[32] L. A. Williams and R. R. Kessler, "All I really need to know about pair programming I learned in kindergarten," *Commun. ACM,* vol. 43, pp. 108-114, 2000.

[33] L. A. Williams and R. R. Kessler, "The effects of "pair-pressure" and "pair-learning" on software engineering education," in *13th Conference on Software Engineering Education & Training, 2000. Proceedings.*, Austin, TX, 2000, pp. 59-65.

APPENDICES

Appendices A and B contain the interview protocols used in Study 2. Appendices C, D, and E contain the questions we asked students in Study 3.

Appendix A. Interview Protocol for Beginning Interviews for Study 2

Below is the interview protocol for participants in the beginning interview for our second study.

SAY: We are evaluating the software tools used in your software engineering course. It is important for the instructors to know how well these new tools are working for you. This is why we want to ask some of the students what they think about the methods in this course.

1. What do you like and dislike most about computer science?

2. What do you like and dislike the most about this course so far?

3. What do you think of the assignments so far?
   a. Can you explain why you think that?
   b. Can you give me an example?
   c. Do you feel like you've accomplished a lot?

4. How strong in the subject do you feel you are compared to your peers?
   a. When faced with a problem or challenge in a course, how confident are you that you can handle it?
   b. What do you feel are your strengths in a group setting?
   c. What do you feel are your weaknesses in a group setting?

5. Have you pair programmed with others in previous classes? At work?
   a. If so, did you enjoy it?
   b. Did it help you complete your task faster?
   c. Do you think you would pair more often if you were able to work with your partner while you were in separate locations?

6. Do you prefer working in pairs/groups or alone?
   a. Why is that true?
   b. Any other reasons?
   c. Do you find that one method is a more effective instructional tool than the other?
   d. What makes pair programming/collaboration an effective tool for you? Why?
   e. What sorts of things are you learning through pairing?

7. What is your class year and what do you see yourself doing after graduation?
   a. Why?
   b. Have you ever done this before? How did it go?
   c. How do you envision the workplace?

d. What do you hope to do in IT? Why are you going in another direction?
e. Do you know people who do what you plan to do?

8. How do you feel about working long hours?

9. What do you do in your spare time?

10. Can you describe to me your image of the computer geek stereotype?
    a. Do you think that is a widespread phenomenon among computer science students?
    b. Where do you see yourself in that stereotype?

Thanks very much for your time and thoughts. Your ideas will help many students here at NC State and in other programs across the United States. The best of success with the semester.

Appendix B. Interview Protocol for Ending Interviews for Study 2

Below is the interview protocol for participants in the ending interview for our second study. Questions in bold font were additions from the beginning interview (Appendix A).

SAY: We are evaluating the software tools used in your software engineering course. It is important for the instructors to know how well these new tools are working for you. This is why we want to ask some of the students what they think about the methods in this course.

1. What do you like and dislike most about computer science?

2. What do you like and dislike the most about this course?

3. What do you think of the assignments so far?
   a. Can you explain why you think that?
   b. Can you give me an example?
   c. Do you feel like you've accomplished a lot?
   d. What was the project like? Did you enjoy the problem? How did it compare to problems in previous courses?

4. How strong in the subject do you feel you are compared to your peers?
   a. When faced with a problem or challenge in a course, how confident are you that you can handle it?
   b. What do you feel are your strengths in a group setting?
   c. What do you feel are your weaknesses in a group setting?

5. **[This question for those who weren't interviewed at the beginning.]** Have you pair programmed with others in previous classes? At work?
   a. If so, did you enjoy it?
   b. Did it help you complete your task faster?
   c. Do you think you would pair more often if you were able to work with your partner while you were in separate locations?

6. **Did you do any pair programming during the team project?**
   a. **If so, did you enjoy it?**
   b. **If so, why did you choose to do so?**
   c. **If so, did it help you complete your task faster?**
   d. **If so, did you work from separate locations or together?**
   e. **If not, why did you choose not to?**

7. So some of the assignments have been paired and some have been solo. Do you prefer working in pairs/groups or alone?
    a. Why is that true?
    b. Any other reasons?
    c. Do you find that one method is a more effective instructional tool than the other?
    d. What makes pair programming/collaboration an effective tool for you? Why?
    e. What sorts of things are you learning through pairing?

8. **Do you feel you have more friends in the department after taking this course? Why or why not?**

9. **Do you feel this course has increased your computer science skill set? Why or why not?**

10. What is your class year and what do you see yourself doing after graduation?
    a. Why?
    b. Have you ever done this before? How did it go?
    c. How do you envision the workplace?
    d. What do you hope to do in IT? Why are you going in another direction?
    e. Do you know people who do what you plan to do?

11. How do you feel about working long hours?

12. What do you do in your spare time?

13. Can you describe to me your image of the computer geek stereotype?
    a. Do you think that is a widespread phenomenon among computer science students?
    b. Where do you see yourself in that stereotype?

Thanks very much for your time and thoughts. Your ideas will help many students here at NC State and in other programs across the United States. The best of success with the semester.

Appendix C. Pre-Study Survey Questions for Study 3

For questions Q5 and Q6, participants were asked to rate their agreement with the statement on a 5-point Likert scale from extremely unlikely to extremely likely. For questions Q7 and Q8, participants were asked to rate their agreement with the statement on a 5-point Likert scale from strongly disagree to strongly agree.

#	Question
Q1	Age
Q2	Gender
Q3	Ethnicity o Caucasian/White o African American/Black o Asian/Pacific Islander o Native American/Native Alaskan o Hispanic/Latino o Rather not say o Other
Q4	Approximately how long does it take you to get to campus from your home? *Please answer in minutes.*
Q5	How likely are you to voluntarily choose to pair program side-by-side with your partner for assignments in this course?
Q6	How likely are you to voluntarily choose to pair program remotely with your partner for assignments in this course?
Q7	I enjoy working with others to accomplish a goal.
Q8	I prefer to work alone when programming.
Q9	Please describe any decisions you and your partner have made about what you will be working on during today's session. *For example, you may have decided to work on a particular requirement or already determined the design you will be following.*

Appendix D. Post-Study Survey Questions for Study 3

For questions Q10 and Q11, participants were asked to rate their agreement with the statement on a 5-point Likert scale from extremely unlikely to extremely likely. For questions Q12-Q18, participants were asked to rate their agreement with the statement on a 5-point Likert scale from strongly disagree to strongly agree.

#	Question
Q10	How likely are you to voluntarily choose to pair program side-by-side with your partner for assignments in this course?
Q11	How likely are you to voluntarily choose to pair program remotely with your partner for assignments in this course?
Q12	I enjoyed pair programming remotely with my partner.
Q13	The distributed pair programming plugin was easy to use.
Q14	The quality of the code we produced during this session was higher than if I had programmed alone.
Q15	The quality of the code we produced during this session was higher than if I had pair programmed with my partner in person.
Q16(a)	I produced about the same amount of code during this session as I would if I had been programming alone.
Q16(b)	I produced about the same or more code during this session as I would if I had been programming alone.
Q17(a)	I produced about the same amount of code during this session as I would if I had been pair programming with my partner in person.
Q17(b)	I produced about the same or more code during this session as I would if I had been pair programming with my partner in person.
Q18	My partner helped keep me on task.
Q19	Please list any benefits or drawbacks to distributed pair programming.
Q20	Please list any recommendations you would give to others trying distributed pair programming.
Q21	What would improve your distributed pair programming experience? *For example, this could be an enhancement to the plugin or an additional tool or communication feature.*
Q22	Other comments

Appendix E. Follow-Up Email Questions for Study 3

For questions Q25 and Q26, participants were asked to rate their agreement with the statement on a 5-point Likert scale from strongly disagree to strongly agree.

#	Question
Q23	Did you pair program remotely again with your partner on homework 3 after the study? Why or why not?
Q24	Did being able to work remotely with your partner simplify scheduling issues? Why or why not?
Q25	I enjoy pair programming by myself.
Q26	I enjoy pair programming in the same location as my partner.