Virtual Collaboration through Distributed Pair Programming

Nachiappan Nagappan, Prashant Baheti
Dr. Laurie Williams
Department of Computer Science
North Carolina State University
Raleigh, NC 27695
+1 919-836-0075
nnagapp@unity.ncsu.edu
ppbaheti@unity.ncsu.edu
williams@csc.ncsu.edu

Dr Edward Gehringer
Department of Computer Science,
Dept. of ECE
North Carolina State University
+1 919-515-2066
efg@ncsu.edu

Dr David Stotts
Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, NC 27599
+1-919-962-1833
stotts@cs.unc.edu

ABSTRACT

Pair programming is a practice in which two programmers work together at one computer, collaborating on the same design, algorithm, code or test. Studies have shown that pair programmers produce higher quality code in essentially the same amount of time as solo programmers. Additional benefits include increased job satisfaction, improved team communication, and efficient tacit knowledge sharing. However, it may not always be possible for all team members to be collocated due to the rise in teleworking and geographically distributed teams. This paper analyzes the results of two distributed pair programming experiments administered at North Carolina State University. Experiment participants used readily available off-the-shelf applications for collaborative development. The results of the experiments indicate that virtual collaboration by distributed pair programmers is on par with collocated collaboration.

Keywords

Collocated teams, distributed teams, pair programming, collaborative programming, distance education, virtual teams, virtual collaboration, Internet Communities.

Technical Area

Collaboration Technology: groupware and Internet communities (virtual collaboration spaces)

1. INTRODUCTION

Distributed team projects are becoming more common in the software industry. The power of distributed development can increase an organization's opportunities to win new work by opening up a broader skill and

product knowledge base, coupled with a deeper pool of potential employees [3]. Major corporations have launched global teams with the expectation that technology will make virtual collocation a feasible alternative [2].

Additionally, distance education (DE) has also come into prominence in recent years. Team projects in DE computer science courses call for distributed development. These teams need to communicate and work effectively and productively. Through the vehicle of groupware, team members can communicate with each other and complete their projects even when they are remotely located or when they work at incompatible hours.

Previous research [1, 4] has indicated that pair programming is better than individual programming in a collocated environment. Do these results also apply to distributed pairs? It has been established that distance matters [2]; face-to-face pair programmers will most likely outperform distributed pair programmers in terms of sheer productivity. However, the inevitability of distributed work in industry and education calls for research in determining how to make this type of work most effective. This paper discusses initial results of research in utilizing the pair programming practice for making distributed software development more effective.

The rest of the paper is organized as follows. Section 2 describes previous work done with respect to pair programming, virtual teams, distributed software development, asynchronous collaboration and extreme programming. Section 3 gives the hypotheses for which we test our results. Section 4 presents the feasibility study for the larger project. Section 5 outlines the experiments that were conducted in graduate classes at North Carolina State University (NCSU). Section 6 presents the results of these experiments. Section 7 outlines the lessons extracted from the experiments. The conclusions and limitations are presented in Section 8.

2. BACKGROUND

In this section, we provide background information on pair programming, virtual teaming, Extreme Programming (XP), distributed software development and asynchronous software development.

2.1 Pair Programming

Pair programming has been practiced sporadically for decades, but has recently been popularized as a practice of the XP methodology [9]. Pair programming is a style of programming in which *two* programmers work side by side at *one* computer, continuously collaborating on the same design, algorithm, code or test. One of the pair, called the *driver*, types at the computer or writes down a design. The other partner, called the *navigator*, has many jobs. One of the roles of the navigator is to observe the work of the driver, looking for tactical and strategic defects in the work

of the driver. Tactical defects are syntax or typographical errors, calls to the wrong method, etc. Strategic defects are said to occur when the team is headed down the wrong path — what they are implementing will not accomplish what it needs to accomplish. Any of us can be guilty of straying off the path. A simple, "Can you explain what you're doing?" from the navigator can serve to bring the driver back onto the right track. The navigator generally has a more objective point of view and can think more strategically about the direction of the work. The driver and navigator can brainstorm on demand at any time. An effective pair-programming relationship is very active. The driver and the navigator communicate at least every 45 seconds to a minute. It is also very important for them to switch the roles of driver and navigator periodically. Note that pair programming includes all phases of the development process — design, debugging, testing, etc. — not just coding. Experience shows that programmers can benefit from pairing at any time during development, in particular when they are working on something that is complex [1,9]: the more complex the task, the greater the need for two brains.

Research has shown that collocated pairs finish in about half the time of individuals and produce higher quality code. The technique has also been shown to assist programmers in enhancing their technical skills and to improve team communication. Approximately 95% of pair programmers surveyed enjoy pair programming more than solo programming and feel more confident in their work when pair programming [1, 9-11].

2.2 Virtual Teaming

A virtual team can be defined as a group of people who work together towards a common goal but operate across time, distance, culture and organizational boundaries [15]. The members of a virtual team may be located at different work sites, or they may travel frequently and need to rely upon communication technologies to share information, collaborate, and coordinate their work efforts. As the business environment becomes more global and businesses are increasingly in search of more creative ways to reduce operating costs, the concept of virtual teams is of paramount importance [6]. In the context of this paper, the common goal of the virtual team is the development of software.

Virtual teams are also used in education. Distance education may be defined as "a form of education in which there is normally a physical separation between teacher and learner. Other means are used to bridge the physical gap — Such as the printed and written word, the telephone, computer conferencing or teleconferencing". [14] .The

concept of virtual teaming is a boon for distance education as it allows distance-learning students to participate in team projects, although the individual team members are geographically dispersed.

Distributed learning, or distance education, is experiencing explosive growth. "Online learning is already a $2 billion business; Gerald Odening, an analyst with Chase Bank, predicts that the figure will rise by 35% a year, reaching $9 billion by 2005" [12]. The federal government assigns great importance to advances in distributed learning. In November 1997, the Department of Defense (DoD) and the White House Office of Science and Technology Policy (OSTP) launched the Advanced Distributed Learning (ADL) initiative. The role of the ADL is "to ensure access to high-quality education and training materials that can be tailored to individual learner needs and made available whenever and wherever they are required" [13]. Programming students have been major participants in the growth of distance education.

It is important to meet the same learning objectives in distance learning as in a traditional classroom. Software project courses, particularly team projects, provide a significant challenge to geographically separated students. Advancements in technology and the invention of groupware have made collaborative programming possible. "Students can now work collaboratively and interact with each other and with their teacher on a regular basis. Students develop interpersonal and communication skills that were unavailable when working in isolation" [16].

With the rise in distance education, teleworking, and globally distributed teams have come publications reporting on effective techniques for managing virtual teams. These have enumerated many challenges and guidelines for virtual teaming. Based on our experiences with the difference between co-located solo programmers and co-located pair programmers, we surmise significant benefits for virtual teams that use distributed pair programming. Distributed pair programming can help to establish team trust and to create a "virtual culture [24]".

A primary consideration in virtual teaming is that of communication [7]. Poor communication can cause problems like inadequate project visibility, wherein everyone does his/her individual work, but no one knows if the pieces can be integrated into a complete solution. Coordination among the team members could also be a problem. Finally, the technology used must be robust enough to support distributed development with ease.

When programmers pair with each other, and especially when the pairs rotate among the group, they get a chance to get to know many on their team more personally. This familiarity helps to break down many communication barriers. Team members find each other much more approachable. They will struggle with questions or lack of information for less time before asking the right person a question — because they know that

person quite well. Additionally, they feel better about their jobs because they know their teammates on a more personal level. Through personal interaction and sharing, they gain knowledge that extends far beyond "book learning." In short, members become more effective.

2.3 Distributed software development

Distributed software development has become more common over the past few years. Software companies practicing distributed software development are on the rise as this saves substantial time and money for the company.

The following factors affect the success of distributed development:

- Distance separating the developers. If the developers are close to each other geographically, then it is possible for them to meet face-to-face occasionally. Otherwise, it may be impossible to meet personally even once.
- Time zone difference. It would be difficult for development teams spread across continental Americas and Asia to work synchronously (as is necessary for distributed pair programming) due to time-zone differences. Much of their work would be done asynchronously with some communication between "shifts."
- Culture. When developers belong to different cultures and speak different languages, effective communication can be particularly challenging.
- Broadband availability. Some countries still have only dialup access. In other countries faster connectivity such as offered by T1, DSL and ISDN may be prevalent.
- Scale/size of the project. If the project is large with several modules, each of which is developed distributedly, then a higher degree of coordination at the management and administrative level is required in order for all the groups to work coherently.
- Location of the customer. This includes proximity of the customer and presence of a customer on-site.
- Corporate culture and political constraints. For example, "In our North American culture, we often expect team members in problem-solving mode to offer direct comments about their opinions and concerns. This is not always true for people who live in other cultures." [23]

In distributed software development, it is necessary to overcome the following obstacles:.

- Shared Mental Model. The idea in the developers mind should be the same, otherwise it made lead to lack of coordination and chaos during the final stages of development.
- Keeping the passion. Generally in distributed programming environments, the passion for development tends to decrease over time. Studies at SMART Technologies Inc., a leading company that develops the award winning SMART Board and other interactive whiteboards, whiteboard cameras and software that facilitates meeting, teaching and training, show "that it is always easiest to work on a project where people are co-located. If someone with whom you work is not walking the same halls as you and not running into you for casual conversation from time to time, then you reduce the camaraderie that is needed for high-performance work teams." [23]
- Communicating with customer. As with collocated teams, communication mechanisms need be put in place for effective communication with the customer and for the dispersion of customer requirements and feedback throughout the virtual team.

2.4 Asynchronous software development

Pair programming is an emerging, but not yet mainstream, practice. Therefore, much of collocated team development takes place asynchronously as programmers work on their own task in their own office/cubicle. Collaboration between collocated team members occurs when the need arises; this on-demand collaboration can take place fairly easily if team members are only steps away from each other. Distributed software teams also often operate in such an asynchronous manner. However, on-demand collaboration can be very difficult for distributed teams. Additionally, as discussed above, asynchronous development might be necessary with distributed teams because of time zone differences.

Development in an asynchronous environment can begin in two ways. The developers decide to split the workload or agree to break down the project into smaller modules and work on these projects together. Each programmer codes independently and mails or checks in/uploads his or her code periodically. This type of development has disadvantages. For example, a break down in communication might make it possible that another person's code is already performing the required operation. Also in asynchronous environments, when a doubt arises about the design then the programmers tend to exchange a flurry of emails which could have otherwise been easily avoided buy a synchronous setup. Perhaps the email exchange might involve a time delay; during this time

developers might make incorrect decisions without proper information in order to make further progress on the project.

Common tools for distributed software development include Microsoft™ SourceSafe, CVS (version control software) etc. These software have a common storage area wherein developers can upload their code and view other developers code too along with details like the last modified date, time, version etc.

2.5 Extreme Programming

Problems in current software development include risk factors such as project cancellation, misunderstood requirements, business and technology changes, and staff/programmer turnover. Extreme Programming (XP) [5] is an agile software development methodology that addresses these risks at all levels of the development process. XP is professed to be efficient, low-risk, flexible, predictable, and a fun way to develop software. First, XP identifies the four basic activities in software development namely coding, testing, listening and designing [5]. XP, used most often for moderately sized projects, specifies 13 core practices for practicing agile software development. These practices are Whole Team, Planning Game, Small Releases, Metaphor, Simple Design, Test-Driven Development, Design Improvement, Pair Programming, Collective Code Ownership, Continuous Integration, Sustainable Pace, Customer Tests and Coding Standards [17]. Pair programming is a key part of XP and is one of the more frequently talked about features.

3 HYPOTHESES

In the fall semester of 2001 and the spring semester of 2002, two experiments were run at NCSU and the University of North Carolina – Chapel Hill (UNC-CH) to assess whether geographically distributed programmer's benefit from using technology to collaborate synchronously with each other. Specifically, we examined the following hypotheses for these experiments:

1. Distributed teams whose members pair synchronously with each other will produce equal or higher quality code than distributed teams that do not pair synchronously.
2. Distributed teams whose members pair synchronously will be more productive (in terms of LOC/hour) than distributed teams that do not pair synchronously.
3. Distributed teams who pair synchronously will have comparable productivity and quality when compared with collocated teams.

4. Distributed teams who pair synchronously will have better communication and teamwork within the team when compared with distributed teams that do not pair synchronously.

4 FEASIBILITY STUDY

4.1 Description

A feasibility study was done in early fall 2001 between NCSU and UNC-CH. The purposes of this study were twofold: (a) to assess a team's satisfaction with collaborating virtually over the Internet utilizing the pair programming practice and (b) to determine an effective technical platform to allow remote teaming. Two pairs of programmers worked over the Internet to develop a modest Java gaming application (a GUI-based Mancala game). Each pair was composed of one programmer from each school, forming a four-person virtual team.

Because the students were only a 40-minute drive apart, they chose to have one face-to-face meeting at the start of the project. In this meeting, the students met each other (for the first time) and discussed an initial design for the project. The students found this meeting highly desirable and recommend that virtual teams do meet face-to-face at least once, if possible. All in all, the students found virtual teaming utilizing a technology solution for distributed pair programming satisfactory. Students who had never practiced collocated pair programming enjoyed the continuous collaboration very much and learned a great deal from each other. (Note: One of the four students had practiced collocated pair programming in a previous class. He had a strong preference for solo programming after this experience; previous results say that 5% of the programmer population will retain a preference for solo programming [1, 9]. This particular student tolerated but did not enjoy distributed pair programming. He found distributed pair programming no worse than collocated pair programming.) A high-quality Mancala game was completed on time.

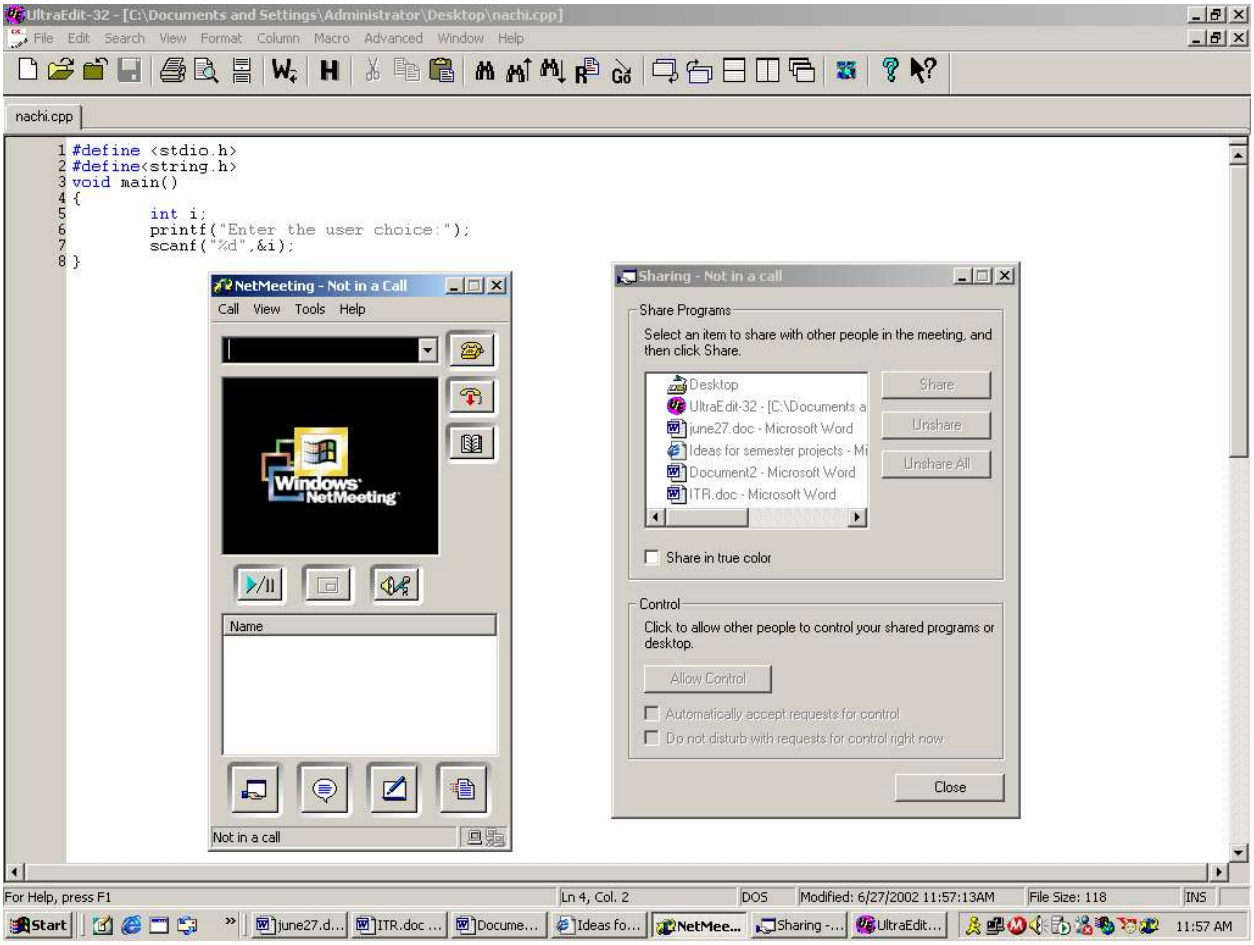
4.2 Collaborative Environment

For collaborating over the Internet, we set out to choose an off the shelf solution that was affordable and easy to learn and use. Ultimately, the following configuration was used for the feasibility study:

4.2.1 Desktop Sharing

First, the team tried pcAnywhere (through a generous donation from Symantec) for desktop sharing. But the team members had trouble passing through each other's firewall as they worked from different universities. Ultimately, the team chose NetMeeting. NetMeeting is a product of Microsoft™ Corporation. "Using your PC and the Internet,

you can now hold face-to-face conversations with friends and family, and collaborate with co-workers around the world"[19]. The interface of Microsoft™ NetMeeting is as shown below.



There are several features of NetMeeting™ that makes it a popular tool of virtual collaborators over the virtual collaborative space. Some of the popular features are now described:

1. *Video and audio conferencing:* This feature enables users to communicate with anyone across the Internet.
2. *Whiteboard:* The Whiteboard enables users to collaborate in real time with others via visual information.
3. *Chat:* This enables users to conduct real-time conversations via text, with multiple users.
4. *File transfer:* File transfer enables developers to send one or more files in the background during a NetMeeting conference.

5. *Program sharing:* This feature enables users to share multiple programs during a conference and also retain greater control over the way they are used.
6. *Remote desktop sharing:* Remote desktop sharing allows users to operate a computer from a remote location, making it possible for users to share the desktop, including the files of the other programmer.
7. *Security:* NetMeeting™ uses three types of security measures to protect privacy namely (i) Data encryption for transferring files, (ii) User authentication for verifying the identity of participants by requiring authentication certificates, (iii) Password protection for starting a conference and for desktop sharing.

4.2.2 Web cams

Web cams can play an important role in virtual collaboration. Web cams allow people to see each other while working which promotes better understanding by viewing the other's body language. It can also be beneficial for a programmer to draw a quick sketch of a basic architectural model and show it on the screen by holding it up to the Web cam.

Through a generous donation of Pocket PC cameras by Intel™, video support via Web cam was available to the students. However, the teams did not find video necessary and chose not to use it.

4.2.3 Headsets and microphones

Headsets and microphones enable audio communication between the two pairs. Using these, the pair communicated with each other just as if they were collocated.

5 Main Experiments

5.1 Experiment 1

After completing the feasibility study, a structured experiment was conducted in a graduate class, Object-Oriented Languages and Systems,[1] taught by Dr Edward Gehringer at North Carolina State University. The course introduces students to object technology and covers OOA/OOD, Smalltalk, and Java. At the end of the semester, all students participate in a five-week team project. We chose this class for our experiment for the following reasons:

1. The projects were developed using an object-oriented language.

[1] http://courses.ncsu.edu/csc517/common

2. The experiment had to be performed on a class that had enough students to partition into four categories and still have enough teams in each category to draw conclusions.

3. We needed some distance-education participants for the class to make distributed development feasible and attractive.

The aforementioned class had 132 students, 34 of whom were distance learning Video-Based Engineering Education (VBEE) students. Almost all the VBEE students were from the Industry. The VBEE students were located throughout the US, often too far apart for collocated programming or even face-to-face meetings. To demonstrate the academic equivalence between on campus and VBEE students the performance of the VBEE students to on-campus students over a period of two years (three semesters), as shown in Table 1. The VBEE students have a comparable mean final average with the on campus students. This is also done to indicate that the results obtained for on-campus classes and VBEE classes are not skewed.

Semester	On-campus	VBEE
Spring 1999	68.41	69.66
Fall 1999	68.23	65.95
Fall 2000	67.84	60.31
Fall 2001	75.59	74.96

Table 1: Mean Final Average Scores

The team project counted for 20% of their final grade. The on-campus students were given 30 days to complete the project, while the VBEE students had 37. (VBEE students' deadlines are typically one week later than on-campus students', because the VBEE students view videotapes[2] of the lectures, which are mailed to them once a week.) Teams composed of some on-campus and some VBEE students were allowed to observe the VBEE deadline, as an inducement to form distributed teams.

All teams were composed of between two and four students. The students' self-selected their teammates, either in person or using a message board associated with the course. Teams then chose one of the four work environments listed below.

1. Collocated team without pairs (9 groups)

[2]The VBEE program now known as Engineering online is moving from videotape to video servers, but this change is not yet complete.

The first set of teams developed their project in the traditional way. Group members divided the tasks among themselves, and each one completed his or her part. An integration phase followed, to bring all the pieces together.

2. Collocated team with pairs (16 groups)

These students worked in collocated pairs. Pair programming was used in the analysis, design, coding and testing phases. A team consisted of one or two pairs. If there were two pairs, an integration phase followed.

The next two environments consisted of teams that were geographically separated, virtual teams. These groups were either composed entirely of VBEE students, or a combination of VBEE and on-campus students.

3. Distributed team without pairs (8 groups)

The third set of teams worked individually on different modules of the project at different locations. The contributions were combined in an integration phase.

4. Distributed team with pairs (5 groups)

This fourth set of teams developed the project by collaborating in pairs over the Internet. These virtual communities worked together on all phases of development, from analysis through test. If there were two pairs, an integration phase followed.

The pairs in our controlled experiment used headsets and microphones to speak to each other. They viewed a common display using readily-available desktop sharing software, such as NetMeeting[3] (distributed with Windows), pcAnywhere[4], or VNC[5] (freely available). Student teams could choose their desk-sharing application of choice. They also used instant-messaging software like Yahoo Messenger while implementing the project. As in the feasibility study, the students were furnished Intel digital cameras to use as Web cams for videoconferencing. We felt these cameras would be useful for speaking face-to-face or to allow them to show paper design documents to each other. However, as earlier, none of these teams found the need to use the Web cams.

As discussed above, in a classic (collocated) pair programming environment, one programmer, the driver, has control of the keyboard and mouse. The navigator watches the work of the driver and does not have control of these input devises; instead, the navigator is watching the work of the driver, identifying defects, giving suggestions, and being an ever-present brainstorming partner. The distributed environment supported these roles well. A typical session involved two programmers sharing desktops, with the navigator having read-only access while the driver

[3] http://www.microsoft.com/windows/netmeeting/default.asp

[4] http://www.symantec.com/pcanywhere/

[5] http://www.uk.research.att.com/vnc/

edited the design, code, or test. The changes made by the driver were seen in real time by the navigator, who was constantly monitoring the driver's work. They could communicate with each other by speaking over the microphone or via instant messaging.

It is interesting to see the main reasons why a person would like to communicate via instant messaging rather than talking over the microphone is that when the other person may not have a compatible Internet connection, i.e. for example one developer may have a DSL while the other developer may be on a dial-up line. Moreover in certain cases where both the developers do not speak the same language it is always better to type out a conversation rater than speak as it can be potentially misunderstood leading to grave complications later.

In order to record their progress, the students utilized an online tool called Bryce [8], a Web-based software-process analysis system used to manage projects and to record metrics for software development. Bryce was developed at NCSU under the direction of the Laurie Williams. Using the tool, the students recorded data including their development time, lines of code, and defects. Development time and defects were recorded for each phase of the software development cycle, namely, planning, design, design review, code, code review, compile and test. Using these inputs, Bryce calculated values for the metrics used to compare the four categories of group projects.

As said above, the students had between 30-37 days to complete the project. The exact development process to be used was not strictly specified; teams followed their own variations of traditional waterfall, iterative, and incremental processes. The projects were implemented in Java. Over the course of the project, the metrics recorded by the students were monitored by the research team so as to make sure that they were recorded on time and were credible. It was found that defects had not been recorded properly by many of the groups, and hence, defects recorded were not considered in this analysis. Two groups (one in category 2 and one in category 3) that had not recorded metrics properly were excluded from the analysis.

A few example projects that were used were calculating and reporting grades in a Peer Grading (PG) program, Java Server Pages program for PG student interface, an assignment-creation wizard for PG, and a GUI for time-tracking tool. Also, it was ensured that the projects were all comparable is terms of lines of code

The two metrics used for the analysis were *productivity*, in terms of lines of code per hour; and *quality*, in terms of the grades obtained by the students for the project. Additionally, after the students had completed their projects, they filled out a survey regarding their experiences while working in a particular category, the difficulties they faced, and the things they liked about their work arrangement.

5.2 Experiment 2

In the spring 2002 semester, eight graduate students, four at NCSU and four at UNC-CH, were selected to participate in a five-week distributed Extreme Programming experiment. These eight students formed four distributed pairs.

Two pairs worked as distributed pairs (utilizing distributed pair programming) and the remaining worked as traditional virtual teams (no pair programming); each pair was assigned to be distributed or paired without considering their preference. All the teams had to conform to the 13 XP practices (except the two traditional virtual teams did not practice pair programming). Each pair worked independently on a card game (four separate versions of the game were produced). The distributed pairs chose to use NetMeeting and Yahoo Messenger for communication whereas the virtual teams wrote the code independently and mailed it back and forth. All four pairs had a common storage area where they could upload their code after modifications and view the other programmer's code. The programming language was Java. JUnit[6] testing was used for all projects. Final results were based on the analysis of the feedback and project output throughout the semester. The number of test cases passed was the metric used for analyzing the quality of the programs.

6 RESULTS

Data was analyzed in terms of productivity and quality, as defined above. Also, qualitative student feedback formed an important third input for the experiment. In the first experiment, our goal was not to show that distributed pair programming is superior to collocated programming (pairing or otherwise). Our goal was to demonstrate that distributed pairing is a viable and desirable alternative. In the second experiment, our goal was to investigate pair programming in a distributed XP environment.

6.1 Experiment 1: Productivity

Productivity was measured in terms of lines of code per hour. Average lines of code per hour for the four environments are shown in Figure 1.

[6] See junit.org.

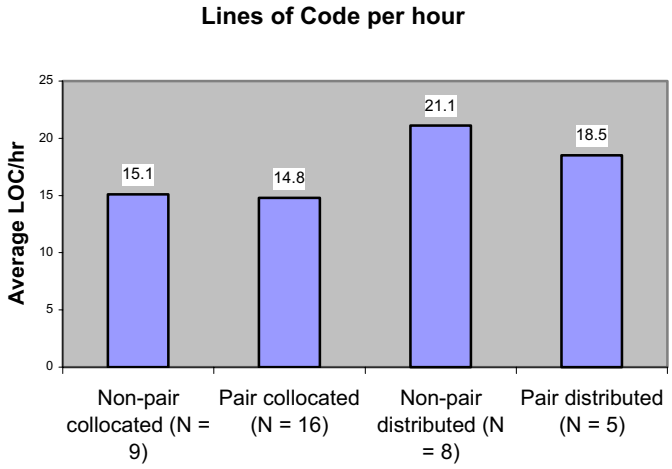


Figure 1: Productivity Bar Chart

The results show that distributed teams had a slightly greater productivity as compared to collocated teams. These results are initially surprising, however, the *f*-test for the four categories shows that results are not statistically significant, due to high variance in the data for distributed groups. If the comparison is restricted to the two distributed categories, a statistical *t*-test on the two categories shows that this difference is not statistically significant at 5% level of significance ($p < 0.74$). In terms of productivity, the groups involved in virtual teaming (without pairs) is not statistically significantly better than those involved in distributed pair programming. (Note: Many are concerned that putting two people on a task one can do via pair programming will significantly hurt productivity. These experimental results are consistent with previous pair programming results [1, 9-11] that indicate that pair programming is not a negative impact on productivity.)

6.2 Experiment 1: Quality

The quality of the software developed by the groups was measured in terms of the average grade obtained by the group out of a maximum of 110. Figure 2 indicates that, again the distributed teams outperformed the collocated teams, but the performance of students did not vary much from one category to another. Although nothing statistically significant can be said about the grades for the four categories, it is interesting to see that those teams performing distributed pair programming were very successful in comparison to other groups. The results of the statistical tests indicate that teams involved in virtual teaming were not significantly better ($p < 0.41$) than the distributed teams using pair programming, in terms of grade. Also, there were no statistically significant results in the comparison between distributed teams and collocated teams, whether paired or not. (Note: The collocated pairs

outperformed the collocated non-pair team; the distributed pairs outperformed the distributed non-pairs. However, the quality improvement was not statistically significant. These results are not consistent with previous pairing results [1, 9-11], which demonstrated a statistically significant quality improvement for pairing.)

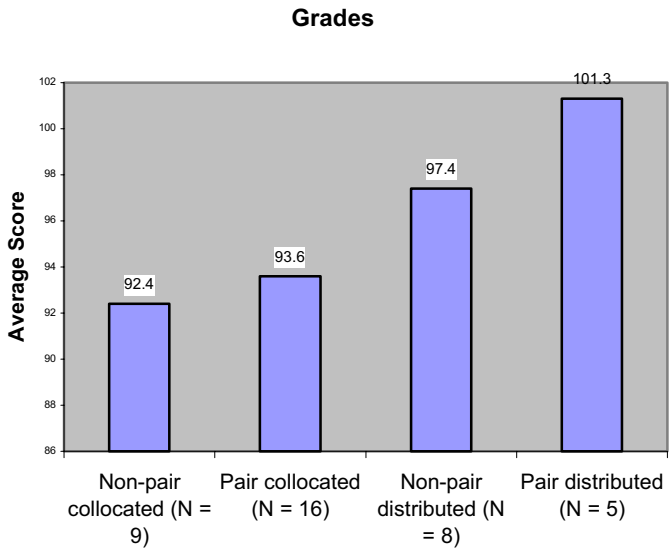


Figure 2: Quality Bar Chart

6.3 Experiment 1: Qualitative Study

Productivity and product quality is important. However, managers and educators strive to provide positive working and learning experiences and environments. We ran a survey to assess participants' satisfaction with their working arrangement. One of the questions was about cooperation within the team. Table 2 shows the responses of the students in the different environments. The pair distributed teams had the best cooperation, followed by pair collocated. This is consistent with prior finding on the benefits of pair programming to team cooperation [1, 9-11]. The non-pair distributed teams reported the worst cooperation (though still not too bad).

	Very Good	Good	Fair	Poor
Non-pair collocated	46%	40%	11%	3%
Pair collocated	62%	28%	10%	0%
Non-pair distributed	45%	37%	18%	0%
Pair distributed	83%	17%	0%	0%

Responses to the question, "How was the cooperation between your team members?"

Table 2: Cooperation within team

The communication among the team members is another important issue in team projects. Table 3 shows the responses of students regarding communication among team members. Again, the distributed pairs fared reported the best communication, followed by the collocated (pair and non-pair) teams. This is also consistent with previous findings on the benefits of pairing on team communication [1, 9-11].

	Very Good	Good	Fair	Poor
Non-pair collocated	57%	26%	11%	6%
Pair collocated	58%	28%	12%	2%
Non-pair distributed	41%	41%	14%	4%
Pair distributed	67%	33%	0%	0%

Responses to the question, "How was the communication with your team?"

Table 3: Communication among Team Members

Over 80% of the distributed pair programming students who answered the survey felt that coding and testing are most suitable phases for distributed pair programming. Collocated pair programmers, in general, found pair programming to be useful in all the phases of software development. When asked to identify the greatest obstacle to distributed pair programming, a few representative students commented as follows:

> *Initially exchanging code/docs via e-mail was a problem. Later on we used Yahoo briefcases to upload code to others to read it from there. From then on things went very smooth.*
>
> *Finding common time available for all.*

The students were asked to identify the biggest benefits of the distributed pair programming, and a few representative comments follow:

> *If each person understands their role and fulfills their commitment, completing the project becomes a piece of cake. It is like Extreme Programming with no hassles. If we do not know one area we can quickly consult others in the team. It was great.*
>
> *There is more than one brain to work on the problem.*
>
> *It makes the distance between two people very short.*

Five out of the six students involved in distributed pair programming thought that technology was not much of a hindrance in collaborative programming when compared with collocated programming.

6.4 Experiment 2: Productivity

In the previous experiment, all student groups implemented different projects. Therefore, our measure of productivity was lines of code/hour. For the distributed XP experiment, all four groups developed the same product.

Therefore, the productivity measure is mean total time for development, which frees the measure from typical concerns with lines of code metrics.

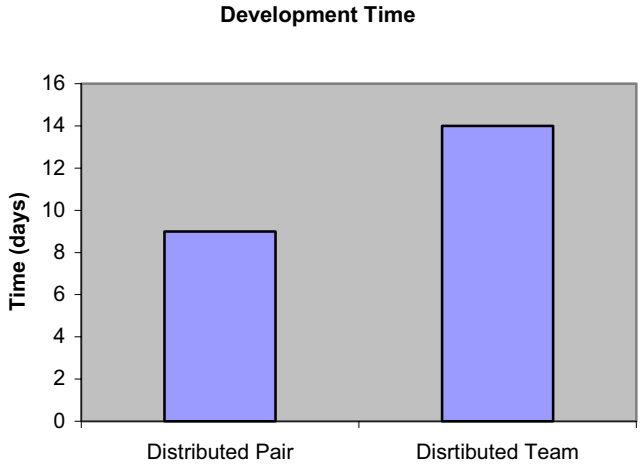


Figure 3: Productivity

Figure 3 shows that the distributed team took a greater amount of time as compared to the distributed pair that worked collaboratively. The distributed teams needed to spend a lot of time coordinating their activities and integrating their code. Whenever a doubt arose, they took more time to clear it due to the limitations of communication. Additionally, previous pair programming research strongly indicates that pairs put a positive form of pressure on each other [1, 9-11]. The pairs made "appointments" with each other for virtual collaboration sessions. The partners always kept their commitments to these appointments and made significant progress during each session. Conversely, the virtual team members often delayed progress because they felt they were "too busy to work on the project right now". Ultimately, this caused a significant delay in project completion. One of the distributed teams never finished the project to completion. One can easily see a parallel between the student work ethic effects of pair programming and a similar arrangement of professional programmers in industry.

6.5 Experiment 2: Quality

Since the programming language used was Java, the teams wrote unit test cases using JUnit. The graph below shows the average number of test cases that written and passed in the JUnit testing of the program.

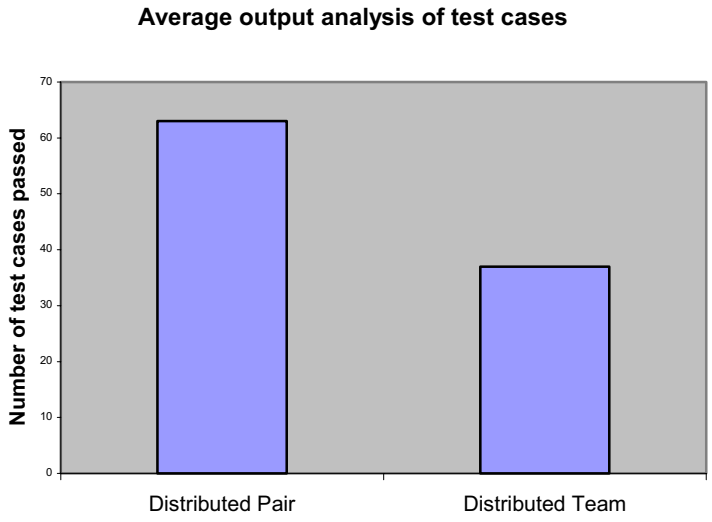


Figure 4: Unit Test Cases Written and Passed

With XP, developers write unit test cases prior to implementing code. In general, we consider that groups that write more unit test cases have better tested code than groups that write fewer test cases. Particularly with XP's test-driven development practice [25], writing more test cases is associated with better structured code that is more likely to ultimately pass acceptance test cases [26]. Figure 4 demonstrates that the distributed pairs wrote 70% more unit test cases test cases than the virtual teams. There could be several reasons for this. First, since the paired teams were working synchronously, they could decide on the flow of the code coherently and decide on the test cases that could be implemented at the same time. The pairs never needed to integrate their code because they worked on the entire project together. The virtual teams needed to integrate their code. At times, they could not write as many test cases to fully test the integration of newly-written code because their partner's code was not yet in the code base. Other significant factors include pair pressure (as discussed above) and pair brainstorming [9]. Pairs are more likely to write a thorough set of test cases because they are "watching over" each other. Lastly, pairs can brainstorm more test cases by putting their brain power together.

The developers were required to record their user stories and the acceptance tests they performed for the software using the same Bryce tool used by the prior experiment. The result of these metrics is shown in Figure 5. These results were obtained by running the code of a fixed set of test cases given in the beginning of the experiment. There were totally 15 tests cases to be satisfied, and both the distributed teams satisfied all. One distributed team satisfied twelve test cases and the other did not complete the project. From Figure 5,, we see that the distributed

teams performed better than the virtual team. But these results were obtained from a very small sample and it is be possible to draw statistically significant results from the above data.

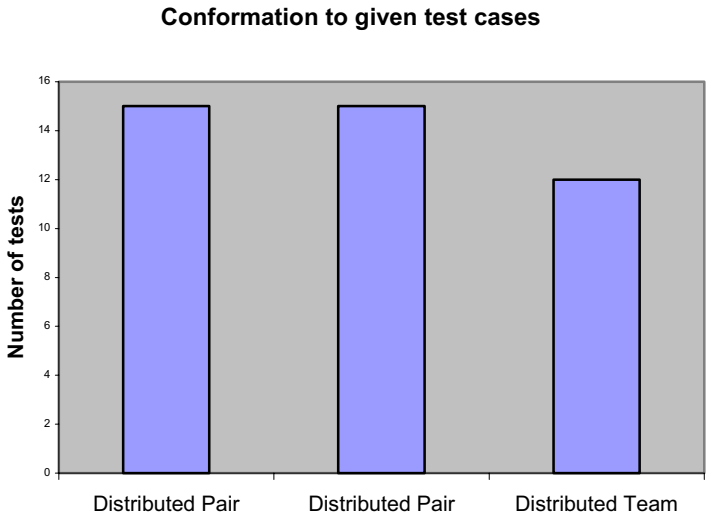


Figure 5: Acceptance tests Passed

6.6 Experiment 2: Qualitative study

Similar to experiment 1, the developers were required to give feedback as to their overall development experience as very good, food, fair and poor. As shown in Table 4, the distributed pairs had a better experience.

Teams	Feedback
Distributed pair 1	Very good
Distributed pair 2	Very good
Distributed team 1	Good
Distributed team 2	Poor

Table 4: Quality of Development experience

The above survey is further supported by the evidence from Figure 5 that shows that the distributed teams finished the project much before the virtual teams.

Also, the results of the survey about the communication between the team members that is an integral part of the software development was found as follows:

Teams	Feedback
Distributed pair 1	Very good
Distributed pair 2	Very good
Distributed team 1	Good
Distributed team 2	Poor

Table 5: Communication among Team Members

The results were found to be exactly similar to the previous case. The main reason for that being that distributed team 2 was not able to complete the project on time due to lack of coordination between the members. Distributed team 1 also experienced certain difficulties when there was a difference in understanding of the architectural model that took almost two days to rectify.

Some of the verbatim comments and experiences are recorded below.

I really enjoyed programming as a pair because there was always an element of peer pressure and I had to accommodate myself when my partner was free as we had to work together. Once we started together there was no looking back and we finished the project much before any of the other teams.

We had a difference of opinion on the basic architectural model and both of us ended up confusing each other. To clarify this doubt we exchanged around 15 emails but finally spoke over the telephone and clarified the doubt in minutes. We spent two days trying to work this problem out.

The distributed team 2 expressed the following:

We both never hit it off from the beginning. There was always some ambiguity involved and a total breakdown of communication in the end. We could not complete our project on time.

From the above qualitative study we can say that in experiment two the distributed teams performed much better than the virtual teams. But we cannot draw any statistically significant results, as the sample size was very small. We intend on repeating this experiment again in the fall semester of 2002.

7 LESSONS EXTRACTED

Based on our feasibility study, we have learned a great deal about designing and operating in an inexpensive, COTS, easy to learn/use virtual software development team environment utilizing pair programming. We summarize these lessons:

1. At least one, but perhaps periodic, face-to-face meeting is beneficial. The students used one such meeting to get to know each other and to brainstorm their initial system architecture.

2. The developers have been found to work better when they strike a good rapport with their partner at a personal level. This was tried here wherein one group in the beginning exchanges the urls to their personal homepage so that one developer could learn about the other.
3. Though we provided Web cam's for the students; they never felt the need to use them. We had surmised that they might periodically use them to talk to each other "face-to-face" or might draw a quick sketch of something and then hold it up to the camera. This need never arose.
4. Using a tool that allows for the distributed teams to quickly switch between a design view, such as a class diagram, and a code view is beneficial. The TogetherSoft Control Center[7] has this capability.
5. Distributed pair programmers absolutely must be willing to speak while they work. They must explain what they are doing as they are doing it or the navigator quickly gets lost. This is so essential that programmers who are not willing to speak almost continuously should probably not try to work this way. Because of this, it is likely that the percentage of the population that can do distributed pair programming will likely be smaller than those that can pair program while collocated with their partner.
6. The virtual teams found out that in certain cases instead of exchanging a number of emails to clear a doubt it may be more efficient to have a direct conversation via telephone as it helps to clarify the doubts faster.

8 LIMITATIONS AND CONCLUSIONS

Our experiments provide initial results of the efficacy of distributed pair programming. The study has some limitations, which we aim to address in future research studies:

- Student teams. Often the external validity or generalizability of empirical studies with students is questioned because student projects do not deal with issues of size or scale, as is reality in industry. Several strong research studies have indicated that student test-beds represent an ideal environment for empirical software engineering, providing sufficient realism while allowing for controlled observation of important project parameters. [21, 22] Additionally, this empirical study involves the interactions between and efficiencies of two programmers working collaboratively. Issues of complexity and scale are not

[7] See http://togethersoft.com

inhibitors to the external validity of a study. We actively seek partners in industry to continue our research in an industrial environment.

- Sample size. For the first experiment the class was large (*N=132*). However, once students were put into teams and teams were assigned to one of four working arrangements, the sample size of groups was not very large. We plan to repeat this experiment in the Fall 2002 semester to build up a larger base of results.

Despite the limitations discussed, our study provides strong initial results of the following findings:

- Pair programming in virtual teams is a feasible way of developing object-oriented software.
- Pair programming in collocated teams is a feasible way of developing object-oriented software.
- Software development involving distributed pair programming seems to be comparable to collocated software development in terms of two metrics, namely productivity (in terms of lines of code per hour) and quality (in terms of the grades obtained).
- Collocated teams did not achieve statistically significantly better results than the distributed teams.
- Feedback from the students indicates that distributed pair programming fosters teamwork and communication within a virtual team.

Thus, the experiments conducted at NC State University are a first indication that distributed pair programming is a feasible and efficient method for dealing with team projects.

ACKNOWLEDGMENTS

We would like to thank NCSU undergraduate student Matt Senter for his help in administering this first experiment. The support of Intel in providing Web cams and Symantec for providing pcAnywhere is graciously acknowledged. We would also like to thank NCSU graduate student Vinay Ramachandran for developing the tool called Bryce to record project metrics and Boby George, Harman Singh, and Sarah Doster for providing valuable feedback regarding this paper.

REFERENCES

[1] L. A. Williams, “The Collaborative Software Process PhD Dissertation”, Department of Computer Science, University of Utah. Salt Lake City, 2000.

[2] G. M. Olson and J. S. Olson, “Distance Matters”. *Human-Computer Interaction*, 2000, volume 15, p. 139–179.

[3] P. E. McMahon, "Distributed Development: Insights, Challenges, and Solutions", *CrossTalk*, http://www.stsc.hill.af.mil/CrossTalk/2001/nov/mcmahon.asp, 2001

[4] J. T. Nosek, "The case for collaborative programming", *Communications of the ACM* 41:3, March 1998, p. 105–108.

[5] K. Beck, "Extreme Programming Explained: Embrace Change". Reading, Massachusetts: Addison-Wesley, 2000.

[6] S. P. Foley, "The Boundless Team: Virtual Teaming", http://esecuritylib.virtualave.net/virtualteams.pdf, Report for MST 660, Seminar in Industrial and Engineering Systems, Master of Science in Technology (MST) Graduate Program, Northern Kentucky University, July 24, 2000.

[7] D. Gould, "Leading Virtual Teams", Leader Values, http://www.leader-values.com/Guests/Gould.htm. July 9, 2000.

[8] http://bryce.csc.ncsu.edu/tool/default.jsp

[9] L. A. Williams, and R. Kessler, *Pair Programming Illuminated*, Boston, MA: Addison Wesley, 2002.

[10] L. Williams, R. Kessler, W. Cunningham, and R. Jeffries, "Strengthening the case for pair-programming", *IEEE Software* 17:4, July/Aug 2000, pp. 19–25.

[11] A. Cockburn, and L. Williams, "The costs and benefits of pair programming", in *Extreme Programming Examined*, Succi, G., Marchesi, M. eds., pp. 223–248, Boston, MA: Addison Wesley, 2001

[12] J. Traub, "This campus is being simulated", *New York Times Magazine*, November 19, 2000, pp. 88–93+.

[13] ADL, "Advanced Distributed Learning", http://www.adlnet.org.

[14] I. Mugridge, "Distance education and the teaching of science", *Impact of Science on Society* 41:4, 1991, pp. 313–320

[15] B. George., Y. M. Mansour, "A Multidisciplinary Virtual Team", Accepted at *Systemics, Cybernetics and Informatics* (SCI), 2002.

[16] M.Z. Last, "Virtual Teams in Computing Education", *SIGCSE 1999: The Thirtieth SIGCSE Technical Symposium on Computer Science Education*, LA, New Orleans, 1999, Doctoral consortium. See page v. of the proceedings.

[17] R. Jeffries, "What is Extreme Programming?" http://www.xprogramming.com/xpmag/whatisxp.htm, November 18, 2001.

[18] R. Jeffries, A. Anderson, C. Hendrickson, "Extreme Programming Installed," Boston: Addison Wesley, 2001.

[19] http://www.fastnloose.com/cgi-bin/wiki.pl/dad

[20] http://www.microsoft.com/windows/NetMeeting/Features/security/default.ASP

[21] A. H. Dutoit, Bruegge, Bernd, "Communication Metrics for Software Development," *IEEE Trans. on Software Eng.*, vol. 24, no. 8, pp. 615-628, 1998.

[22] W. S. Humphrey, *A Discipline for Software Engineering.* Reading, Mass.: Addison Wesley Longman, Inc., 1995.

[23] http://www.effectivemeetings.com/technology/virtualteam/virtualteaming.asp

[24] P. E. McMahon, *Virtual Project Management: Software Solutions for Today and the Future*. Boca Raton: St. Lucie Press, 2001.

[25] K. Beck, *Test-Driven Development*, Boston, Mass.: Addison Wesley, in press.

[26] B. George, *Analysis and Quantification of Test Driven Development Approach*, MS Thesis, North Carolina State University, 2002.