

# Exploring Pair Programming in Distributed Object-Oriented Team Projects

Prashant Baheti  
Department of  
Computer Science  
North Carolina State  
University  
Raleigh, NC 27695  
+1 919-755-1264  
ppbaheti@unity.ncsu.edu

Dr Laurie Williams  
Department of  
Computer Science  
North Carolina State  
University  
Raleigh, NC 27695  
+1 919-515-7925  
williams@csc.ncsu.edu

Dr Edward  
Gehring  
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

Previous research [1, 4] has indicated that pair programming is better than individual programming when the pairs are physically collocated. However, important questions arise: How effective is pair programming if the pairs are not physically next to each other? What if the programmers are geographically distributed? An experiment was conducted at North Carolina State University to compare the different working arrangements of student teams developing object-oriented software. The teams were both collocated and in distributed environments; some teams practiced pair programming while others did not. The results of the experiment indicate that it is feasible to develop software using distributed pair programming, and that the resulting software is comparable to software developed in collocated or virtual teams. Our findings will be of significant help to educators dealing with team projects for distance-learning students, as well as organizations that are involved in distributed development of software.

## Keywords

Extreme Programming (XP), collocated, distributed, pair programming, collaborative programming, distance education, virtual team.

## 1. INTRODUCTION

Distance education (DE) has 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.

Distributed team projects are also very common in the software industry. Employing 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 personnel to potentially employ [3]. Major corporations have launched

global teams, expecting that technology will make "virtual collocation" a feasible alternative [2].

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 DE calls for research in determining how to make this type of work most effective. This is the focus of this paper. We believe that our results also have implications for industry, where virtual teams are quite common.

The rest of the paper is organized as follows. Section 2 describes the previous work done with respect to pair programming and virtual teams. Section 3 gives the hypothesis for which we test our results. Section 4 outlines the experiment that was conducted in a graduate class at North Carolina State University (NCSU). Section 5 presents the results. It is followed by an outline of future work in Section 6. The conclusions are presented in Section 7.

## 2. PREVIOUS WORK

### 2.1 Pair Programming

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*, is 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 errors, typos, 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 won't 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 has a much more objective point of view and can better think 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 roles periodically. Note that pair programming includes all phases of the development process — design, debugging, testing, etc. — not just coding. Experience shows that programmers can pair at any time during development, in particular when they are working on something that is complex: the more complex the task, the greater the need for two brains [1, 9].

Research has shown that 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, to improve team communication, and to be more enjoyable [1, 9, 10, 11].

## 2.2 Virtual Teaming

A virtual team can be defined as a group of people, who work together towards a common goal, but across time, distance, culture and organizational boundaries [15]. In our context the goal is development of software. 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].

DE may be defined as “a form of education in which there is normally a separation between teacher and learner, and thus one in which other means — the printed and written word, the telephone, computer conferencing or teleconferencing, for example — are used to bridge the physical gap” [14]. The concept of virtual teaming is a boon for distance education as it allows distance-learning students participate in team projects, although the individual team members are geographically dispersed.

In the past, there was no support for collaborative programming for virtual teams. Advancements in technology and the invention of groupware have changed this situation. “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].

DE is experiencing explosive growth. “Online learning is already a \$2 billion business; Gerald Odening, an analyst with Chase Manhattan 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. However, software project courses, particularly team projects, provide a significant challenge to geographically separated students.

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.

In the educational field, with distance learning on the rise, virtual teaming has become inevitable. At the same time, it is important to meet the same learning objectives in distance learning as in a traditional classroom.

## 3. HYPOTHESES

In the fall of 2001, we ran an initial experiment at North Carolina State University to assess whether geographically distributed programmers benefit from using technology to collaborate synchronously with each other. Specifically, we examined the following hypotheses:

- Distributed teams whose members pair synchronously with each other will produce higher quality code than distributed teams that do not pair synchronously.
- Distributed teams whose members pair synchronously will be more productive (in terms of LOC/hr.) than distributed teams that do not pair synchronously.
- Distributed teams who pair synchronously will have comparable productivity and quality when compared with collocated teams.
- 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. THE EXPERIMENT

An initial feasibility study was done in early fall 2001 between NCSU and UNC 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; each pair was composed of one programmer from each remote site. From this experiment we found that effective remote teaming could be done with the PC sharing software and audio support we describe in the following section; video support was provided as well, but the teams did not find video necessary and chose not to use it.

After the feasibility study, a formal experiment was conducted in a graduate class, Object-Oriented Languages and Systems,<sup>1</sup>

---

<sup>1</sup><http://courses.ncsu.edu/csc517/common>

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 5-week team project. We chose this class for our experiment for the following reasons:

1. The projects were developed using an object-oriented language.
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. The VBEE students were located throughout the US, often too far apart for collocated programming or even face-to-face meetings. 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<sup>2</sup> 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..

Teams were composed of 2-4 students. The students self-selected their teammates, either in person or using a message board associated with the course, and chose one of the four work environments listed below.

#### 1. *Collocated team without pairs (9 groups)*

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)*

The next set of groups worked in 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 working in pairs over the Internet. At the end, they integrated the various modules.

The pairs in this experiment used headsets and microphones to speak to each other. They viewed a common display using desktop sharing software, such as NetMeeting, PCAnywhere, or VNC. They also used instant-messaging software like Yahoo Messenger while implementing the project. A typical session involved two programmers sharing desktops, with one of the pair (the navigator) having read-only access while the other (the driver) actually edited the code. The changes made by the driver were seen in real time by the navigator, who was constantly monitoring the driver's work. The navigator could communicate with the driver by speaking over the microphone, or via instant messaging. As in the initial platform experiment, the students were furnished Intel digital cameras to use as Webcams for videoconferencing, to allow them, for example, to show paper design documents to each other. However, as earlier, none of these teams found the need to use the Webcams.

In order to record their progress, the students utilized an online tool called Bryce [8], a Web-based software-process analysis system used to record metrics for software development. Bryce was developed at NCSU under the direction of the second author. 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.

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 time metrics properly were excluded from the analysis.

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. RESULTS

Data were analyzed in terms of productivity and quality, as defined above. Also, student feedback formed an important third input for the experiment. Our goal was not to show that distributed pair programming is superior to collocated pair programming for student teams. Our goal was to demonstrate that distributed pairing is a viable and desirable alternative for use with student teams, particularly for distance education

---

<sup>2</sup>The VBEE program is moving from videotape to video servers, but this change is not yet complete.

students. We plan to repeat this experiment in the Fall 2002 semester to build up a larger base of results.

## 5.1 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.

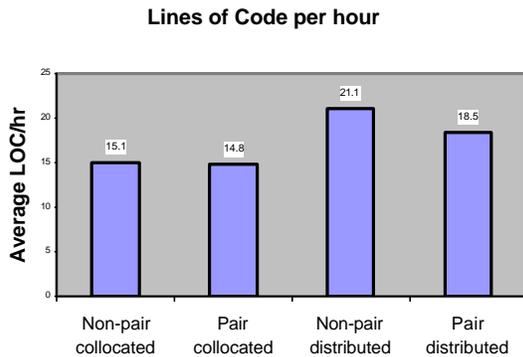


Figure 1

The results show that distributed teams had a slightly greater productivity as compared to collocated teams; however, the *f*-test for the four categories shows that results are not statistically significant ( $p < 0.1$ ), due to high variance in the data for distributed groups. This is better depicted by the box plot (Figure 2) for the four categories, which illustrates the distribution of the metric for the four environments.

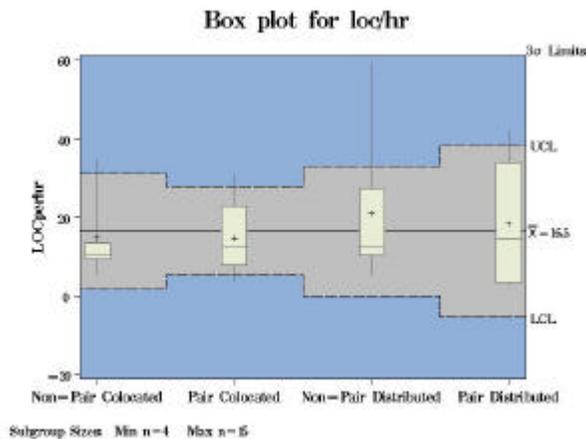


Figure 2

A box plot shows the distribution of data around the median. The vertical rectangle for each category shows the distribution of the middle 50% of the readings. The horizontal line inside each rectangle shows the median value for that particular category. The line segment from the top of the rectangle shows the range in which the top 25% of the values lie. Similarly, the line segment below the rectangle shows the range in which the bottom 25% of the values lie. Thus, the end points of the two line segments indicate the total range that the values for a particular category fall into. For example, the

median for the non-pair collocated category is around 10 LOC/hr., with the middle 50% of the values lying between approximately 9 and 13 LOC/hr., while the entire range is between 5 and 35 LOC/hr., approximately.

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. In terms of productivity, the groups involved in virtual teaming (without pairs) is not statistically significantly better than those involved in distributed pair programming. In other words, teams involved in distributed pair programming perform similarly to those on virtual teams without distributed pair programming.

## 5.2 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. The graph below indicates that the performance of students did not vary much from one category to another.

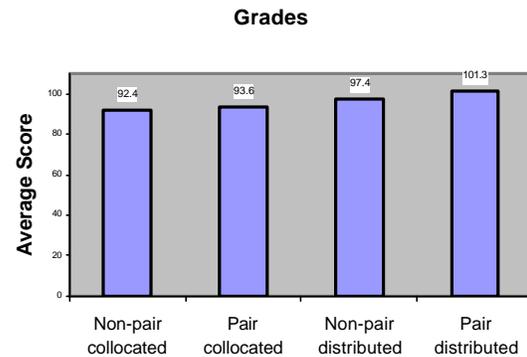


Figure 3

A box plot for the grades only corroborates the claim made above. 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 distributed teams with pair programming performed similarly to those distributed teams that did not practice pair programming in terms of project grade.

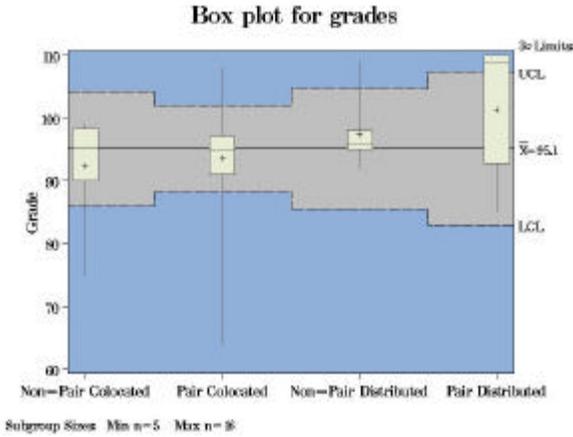


Figure 4

### 5.3 Student Feedback

Productivity and product quality is important. However, as educators we strive to provide positive learning experiences for our students. We ran a survey to assess students' satisfaction with their working arrangement. One of the questions was about cooperation within the team. Table 1 shows the responses of the students in the different environments.

	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 1: Cooperation within team

The communication among the team members is an important issue in team projects. Table 2 shows the responses of students regarding communication among team members.

	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 2: Communication among Team Members

The survey also indicates that five out of six students 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, 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 responded—

"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. Also, 23 out of 28 students involved in virtual teaming with or without pair programming felt that there was proper cooperation among team members.

## 6. FUTURE WORK

The experiment we conducted was a classroom experiment among 132 students, including 34 distance-learning students. To be able to draw statistically significant conclusions, such experiments have to be repeated, on a larger scale if possible. However, this experiment has given initial indications of the viability of distributed pair programming. We intend to conduct more experiments like this so that we can draw conclusions about distributed pair programming, and whether virtual teams should be a standard practice in the classroom as well as in industry.

## 7. CONCLUSIONS

The results of our experiment indicate the following:

- 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 experiment conducted at NC State University is a first indication that distributed pair programming is a feasible and efficient method for dealing with team projects.

## 8. ACKNOWLEDGMENTS

We would like to thank NCSU undergraduate student Matt Senter for his help in administering this experiment. The support of Intel in providing equipment is graciously acknowledged. We would also like to thank NCSU graduate student Vinay Ramachandran for developing the tool called Bryce to record project metrics.

## 9. 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.