Technical Report TR03-003

Department of Computer Science
Univ. of North Carolina at Chapel Hill

Virtual Teaming: Experiments and Experiences with Distributed Pair Programming

David Stotts[1], Laurie Williams[2], Nachiappan Nagappan[2], Prashant Baheti[2], Dennis Jen[1], Anne Jackson[2]

[1]Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599-3175

[2]Department of Computer Science
North Carolina State University
Raleigh, NC 27695

stotts@cs.unc.edu

williams@csc.ncsu.edu

March 1, 2003

Virtual Teaming: Experiments and Experiences with Distributed Pair Programming

David Stotts[1], Laurie Williams[2], Nachiappan Nagappan[2], Prashant Baheti[2], Dennis Jen[1], Anne Jackson[2]

[1]Dept. of Computer Science, University of North Carolina, Chapel Hill, NC 27599
{stotts, dsjen}@cs.unc.edu
[2]Dept. of Computer Science, North Carolina State University, Raleigh, NC 27695
{lawilli3, nnagapp, ppbaheti, amjackso}@unity.ncsu.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. Previous 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 case studies done at UNC Chapel Hill and at NC State University. Participants used readily available off-the-shelf applications for collaborative software development. The results indicate that software development collaboratively "over the wire" is feasible, effective, and pleasant for the participants; distributed development is better done as synchronous pairs than as individuals who integrate; and distributed pairs maintain many of the advantages of collocated pairs.

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 [12]. Major corporations have launched global teams with the expectation that technology will make virtual collocation a feasible alternative [15]. 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 [14,20] has indicated that pair programming is better than individual programming in a co-located environment. Do these results also apply to distributed pairs? It has been established that distance matters [15]; 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. Additionally, Extreme Programming (XP) [3] usually has co-located pairs working in front of the same workstation, a limitation that ostensibly hinders use of XP for distributed development of software.

This paper discusses results of our research on *distributed pair programming (dPP)*. By *dPP* we mean that two members of the team (which may consist solely of these two people) synchronously collaborate on the same design or code from different locations. This means that

both must view a copy of the same screen, and at least one of them should have the capability to change the contents on the screen. To be able to do this, they require technological support for sharing desktops and verbal conversation, and perhaps even video conferencing capabilities.

Our first dPP experiments have been previously reported [1,2]. This paper gives results of two other case studies done jointly between grad students[1] at the University of North Carolina at Chapel Hill (UNC-CH) and grad students at North Carolina State University (NCSU) in the spring and fall of 2002. Section 2 gives background work on virtual teams, a summary of prior dPP results, and a description of the technical infrastructure to support dPP. Sections 3 and 4 discuss the details of the two case studies. Section 5 outlines the lessons extracted from the experiments. General observations, limitations, and conclusions are presented in Section 6.

2. Background and Related Work

Virtual teaming

Our studies involve a specific form of *virtual team*. In general, 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 [8]. 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 [7]. 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. 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" [17]. Programming students have benefited from this growth. Virtual teaming is a boon for distance education as it allows geographically remote students to participate in team projects.

Organizing and managing virtual teams is a topic of ongoing research. From our earlier studies and experiences comparing co-located solo programmers with co-located pair programmers [1,2], we surmise significant benefits for virtual teams that use dPP. Operating via dPP may help establish team trust and create a "virtual culture [13]". 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. As a result, they will struggle with questions or lack of information for less time before asking the right person a question. The rotation of team members also gives each student a broader understanding of the project through observing the work of each new partner. Additionally, they feel better about their jobs because they know their teammates on a more personal level. In short, better communication between team members leads to increased confidence levels and more effective time usage. A primary consideration, then, in virtual teaming (and so dPP) is good support for communication [10].

[1] The validity or generalizability of empirical studies with students is sometimes questioned because student projects do not deal with issues of size or scale, as is realistic in industry. Several research studies have indicated, however, that student test-beds represent ideal environments for empirical software engineering, providing sufficient realism while allowing for controlled observation of important project parameters [6,11].

Prior Distributed Pair Programming Results

In the Fall 2001 semester a structured experiment was conducted in a graduate class, Object-Oriented Languages and Systems, taught by Dr Edward Gehringer at NCSU [1,2]. This course introduces students to object technology and covers object-oriented analysis and design, Smalltalk, and Java. This course has a five-week team project that was used for our experiment. A total of 132 students took this course, including 32 distance education students. For the team project, the students were divided into teams of two to four students and worked as collocated teams, collocated team with pair programming, distributed teams, and distributed team with pairs.

The results of this experiment show that distributed teams had a slightly greater productivity as compared to collocated teams but the difference was not statistically significant. Also the distributed teams outperformed the collocated teams in terms of software quality measured by the average grade obtained by the group in the project. Again, the difference was not statistically significant. Anecdotally, the co-located pairs outperformed the co-located non-pair teams, and the distributed pairs outperformed the distributed non-pairs.

Another area under study is the communication among team members. We measured this with an exit survey. The distributed pairs reported the best communication, followed by the collocated (pair and non-pair) teams. This is consistent with earlier findings on the benefits of pairing on team communication [18,20].

Technical Infrastructure Considerations for dPP

For collaborating over the Internet, we chose COTS solutions that are affordable, readily available, and easy to learn and use. One goal of our work has been to see how effective dPP can be with a simple, non-custom setup. Table 1 lists the programs and technologies used at various times and in various combinations during our two case studies and our previous experiments.

Technology	*Capabilities*	*Comments from users*
NetMeeting	Program Sharing, Whiteboard, Text Messaging, Voice Communication	We continued to refer to the whiteboard stored on NetMeeting.
pcAnywhere	Desktop Sharing	Some firewall issues.
WS FTP	File Transfer	
TextPad	Text Editing	
Notepad	Text Editing	
Crimson Editor	Text Editing	Color codes JSP files.
Visio	Diagram Creating Software	Used to create our web flow diagram.
Yahoo Messenger	Text messaging, File Transfer, Voice Communication	We initially used this, but decided against it because MSN Messenger can start NetMeeting easily, while also having the same features as Yahoo Messenger.
MSN Messenger	Text messaging, File Transfer, Voice Communication, Start NetMeeting	
WinZip	Zip and Unzip Files	
HomeSite	HTML Editor	Used to write the meeting minutes.
CVS	File Repository and Management	It was too difficult to set up and

Technology	Capabilities	Comments from users
		seemed unnecessary.
Tomcat	Server	Allowed us to view JSP.
Internet	Information and Communication Medium	
Group Web Site	Web Site	Used to store meeting minutes and project documents
Internet Explorer	Web Browser	
Netscape	Web Browser	
Email	Text Communication/File Transfer	
Putty	Shell	
Eclipse	Integrated development environments (IDE)	We tested the Sangum plug-in for dPP.

Table 1. Programs and technologies used in our studies

We have developed systems with dPP infrastructures based on both *NetMeeting*™ (Microsoft) and *pcAnywhere* (Symantec). Some pairs have preferred one, some the other; we have recorded the various reasons given for the preferences. Both programs, though, share important functions and characteristics needed for dPP:

> *remote desktop sharing, program sharing, file transfer, session security (password login, authentication, encrypted transmissions)*

In addition, these capabilities are needed (or desirable) for dPP:

> *audio conferencing, whiteboard, text chat*

NetMeeting provides them integrally; but *pcAnywhere* requires third-party programs that are then shared with the desktop. Finally, *video* may have uses in dPP; this is a point for further research.

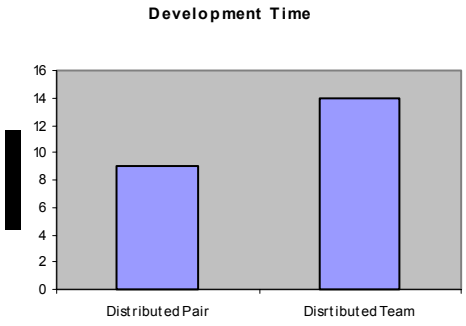
3. Spring 2002 Comparative Study

In Spring 2002, eight graduate students (four at NCSU, four at UNC-CH) participated in a five-week dPP/dXP experiment. We formed four distributed pairs, each having one student at UNC and one at NCSU. About 30 miles separates these locations so there was no face-to-face contact within a group, and all communication was done via the Internet connection between the campuses.

Two of the groups worked as virtual synchronous pairs (utilizing dPP); we refer here to them as "dPP pairs". The remaining two worked as more traditional virtual teams (no pair programming); we refer to them as distributed, non-paired teams, or "dNP teams". Allocation of students to groups was done randomly without regard to preferences. All four groups had to conform to the 13 XP practices (except the two dNP teams did not practice pair programming). Each group worked independently on a card game, so four separate versions of the same game were produced. The dPP pairs first tried *pcAnywhere* for desktop sharing, but they had trouble passing through each other's firewalls as they worked from different universities. Ultimately, they chose *NetMeeting* for development, but *Yahoo Messenger* was favored over *MSN Messenger* for voice communication. The dNP teams wrote their code independently and e-mailed it back and forth. All four groups had a common storage area where they could upload their code after modifications

and view the other programmer's code. Programming was done in Java, and JUnit[2] testing was used for all projects.

The results reported here came from analysis of programmer feedback and project output throughout the experiment. The *number of test cases passed* is the metric used for program *quality*. Since all groups developed the same product, the *productivity* measure is *mean total time* for development; this frees the analysis from typical concerns with lines of code measures.



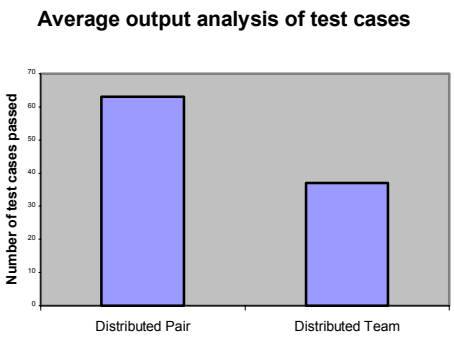


Fig. 1. Development Time (days)

Fig.2. Unit tests written and passed

Figure 1 shows that the dNP teams took a greater amount of time (in days) compared to the dPP pairs. The dNP teams spent considerable time coordinating their activities and integrating their code. Whenever a question arose, they took more time to clear it due to the limitations of communication. The dPP pair members 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 dNP 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 dNP team never finished the project to completion. This supports earlier findings that pairs put a positive form of "pair pressure" on each other [5,18-20]. One can easily see a parallel between the student work ethic effects of pair programming and similar effects among professional industrial programmers.

Since programming was in Java, the groups wrote unit test cases using JUnit. Figure 2 shows the average number of test cases that were written and passed in JUnit testing. DPP pairs wrote just over 60 tests, whereas dNP teams wrote just under 40 tests. XP requires *test-driven development (TDD)*, meaning programmers write unit test cases prior to implementing code [4]. In general, we consider that groups writing more unit test cases have better tested code than groups writing fewer test cases. Particularly with TDD, writing more test cases is associated with producing better structured code that is more likely to ultimately pass acceptance tests [9].

Figure 2 shows that dPP pairs wrote 70% more unit test cases than the dNP teams. Since the dPP pairs were working synchronously, they could concurrently decide on the flow of the code and on the test cases that could be implemented. These pairs never needed to integrate their code because they worked on the entire project together. The dNP teams needed to separately and specifically integrate their individual efforts. 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 and pair brainstorming [18]. Pairs are more

[2] See http://www.junit.org/ .

likely to write a thorough set of test cases because they are continually "watching over" each other and can brainstorm more test cases by putting their brainpower together.

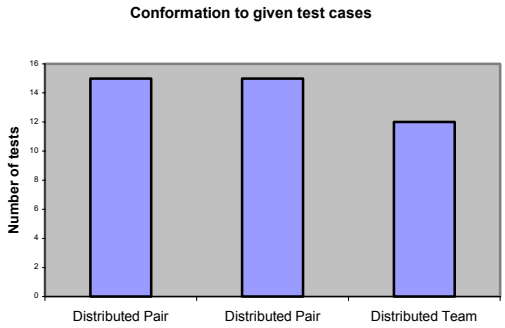


Fig. 2. Acceptance tests Passed

All four groups were required to record their user stories and the acceptance tests they performed for the software using Bryce[3], a Web-based software-process analysis system used to manage projects and to record development metrics. The results of these measures are shown in Figure 3. These results were obtained by running the code against a fixed set of 15 test cases determined before the experiment. Both the dPP pairs satisfied all the test cases. One dNP team satisfied 12 test cases, and the other did not complete the project. The sample size is too small to do a statistically significant analysis of the data.

Teams	*Quality of Development Experience*	*Communication Among Team Members*
dPP pair 1	Very good	Very good
dPP pair 2	Very good	Very good
dNP team 1	Good	Good
dNP team 2	Poor	Poor

Table 2. Qualitative Feedback

The developers were also required to give feedback about their overall development experience and their team communication. The allowed response range was *very good, good, fair* and *poor*. As shown in Table 2, the dPP pairs reported a better experience. The main reason for this was that dNP team 2 was not able to complete the project on time due to lack of coordination between the members. Moreover, dNP team 1 experienced difficulties when there was a difference in understanding of the architectural model that took almost two days to rectify. From this case study we can say we have further suggestive evidence that the synchronous paired teams performed better than the non-paired teams.

4. Fall 2002 Case Study

The second case study we have completed was done in the fall of 2002. We created one pair, distributed with one grad student at UNC and the other from NCSU. The NCSU student worked from a home office, connected to the Internet via cable modem. The UNC programmer used a

[3] See http://bryce.csc.ncsu.edu

campus office with a 100-megabit Internet connection. Unlike the prior comparative study, in this development there was no face-to-face meeting to start the project. Email was used for initial contacts and team organization. Four meetings online were needed over the course of the first three weeks to try various technologies and settle on a collection of tools that worked well for the computing environment the pair members had. Table 3 summarizes the computing environments used by each programmer.

Property	Remote system	Host system
Net Connection	Internet II backbone, UNC office	Cable modem, home office
Communications	Headset	Speakers and microphone
Operating Sys	Windows2000	Windows 98
Ram	128 MB	384 MB
Processor Speed	400 MHZ	933 MHZ

Table 3. Computing Platforms used by the pair members

The pair sessions were divided into two main segments: *infrastructure tests*, and *development*. The first few pair sessions were technology tests, spent trying various combinations of dPP support programs for effectiveness and to establish their preferences. The pair settled on *MSN Messenger* for voice communication and *pcAnywhere* for screen sharing. The first session after that was used to code a simple magic square program as a "development shake down", in which the pair became accustomed to the behavior patterns needed to produce working code in the dPP environment they chose. Once their dPP environment was established, the remaining sessions comprised the measured development. Each development session had six activity blocks:

1. Each person logs onto MSN Messenger.
2. One of the pair would request a voice communication.
3. Start *pcAnywhere* (UNC as remote, NCSU as host)
4. Pair programming
5. Discussion about what to do in the next meeting and confirm next day to meet.
6. Post meeting minutes to website.

The pair produced a tool to support future XP projects: a pair matcher that takes factors such as experience, personality type, and preferences into account to try to form pairs that are likely to be effective. This project ended up as a set of about 20 Java server pages with a web interface. The pair spent a total of 37.25 hours in development from 9/19/02 to 11/25/02, using 18 online sessions averaging 2.07 hours each. The longest session was 3.25 hours, and the shortest was 0.75 hours. Minutes and observations were kept of all meetings; they can be reviewed online at the project web site http://www.cs.unc.edu/~dsjen/pair/ along with user stories for the program and an architecture diagram of the system they produced.

Observations on the dPP technical infrastructure

The software used fir dPP must compensate as much as possible for the lack of physical contact between team members. The team in this study decided that *pcAnywhere* best emulated the co-located environment (the prior study participants used *NetMeeting*). Alternate environments, such as *NetMeeting* and *Eclipse* with a pair programming plug-in[4], also allowed the sharing of programs, but the methods for doing so were deemed less effective for the pair. The following are technical problems observed with the dPP infrastructure:

[4] See http://www.industriallogic.com/software/sangam.html

- Inability to copy and paste from one computer to another. The person connecting to the other's desktop was not able to copy and paste from his own desktop, which would have been a convenient feature.
- In *NetMeeting*, mouse locus behavior prevented use of a PC when the other pair member was driving.
- *NetMeeting* exhibited graphics problems drawing cursors that *pcAnywhere* solved.
- Network-based voice communication occasionally would break up, making hearing one's partner very difficult.
- One partner was using speakers instead of a headset, producing an audible echo in the headset of the partner (who would hear himself talking with a delay). Initially the partner found this distracting, and spoke slowly and haltingly to compensate. However, he reported becoming used to it, could ignore it, and even expected it as an indication of a live connection.
- When using *pcAnywhere*, transferring control to another is much easier.
- Remote machine should have a screen size slightly larger than the host machine; this allows the window showing the host PC to fit entirely on the remote, requiring no scroll bars.
- There was some lag in mouse motion and editor scrolling; however, it was minimal, easily adapted to, and not noticeable after a few initial sessions.

5. Lessons Learned

These new studies, and our earlier ones, have allowed us to gathered some observations we think characterize effective virtual team development of software with dPP using an inexpensive, COTS, easy to learn/use technical environment. These lessons include:

- At least one, but perhaps periodic, face-to-face meeting is beneficial. In the comparative study, the students used one such meeting to get to know each other and to brainstorm their initial system architecture.
- The developers have been found to work better when they strike a good rapport with their partner at a personal level. Groups in the beginning exchanged URLs to their personal Web homepages so that one developer could learn about the other.
- 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*[5] has this capability.
- 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. Programmers who are not willing to speak almost continuously should probably not try to work this way.
- Beyond the necessary basics (screen sharing, audio communications, file transfer), the appropriate technical infrastructure for dPP appears to vary with individual tastes; some teams were forced to one product or another by specific computing platform issues (firewalls, communication speeds), but overall different teams ended up selecting different combinations of *NetMeeting, pcAnywhere, MSN Messenger, and Yahoo Messenger*. All combinations worked effectively once the programmers were happy.
- Screen sharing programs used in dPP alleviate potential file duplication, data coherence and consistency problems that could occur with integrating forms of virtual teaming; one member of the pair is always the host and work is always off one project base.

[5] See http://togethersoft.com

Advantages of dPP over co-located PP

In exit interviews, the participants noted they had benefited from many of the previously observed advantages of co-located pair programming, such as pair learning, pair pressure, two-brains better than one, etc. Our studies indicate that distribution does not destroy or hinder these co-located PP advantages. In addition, distributed PP has these advantages over co-located PP:

- Visibility is improved over collocated pair programming at a single PC/monitor, since each dPP participant has a screen.
- The navigating dPP participant can use the PC to search the Web for resources
- No office changing or travel is needed to meet one's partner; work on other projects can continue until dPP appointment time.
- Although not tested, meetings are possible when on trips, out of town, etc.
- Pairs are forced to keep electronic copies and records of our work and ideas. For example, instead of drawing on a physical whiteboard, the participants used NetMeeting's whiteboard. This ensured they would be able to go back and look at earlier plans.
- Pair members are less likely to start conversations off topic; meetings are almost completely focused on the task. The computer is the medium for all exchanges, and participants can't turn away from their computers and chat one-on-one.

Disadvantages of dPP compared to co-located PP

The study participants observed these disadvantages of dPP over co-located PP:

- Users can't point, making it difficult to describe where a problem is; line number naming helps, but it takes a noticeable amount of time for the other to find the line number.
- A problem with one computer forces both to stop working; this theoretically doubles the MTTF over using a single computer (as in co-located PP)
- Pair members can't see facial expressions; Webcams are too small, too limited in frame rate, and too expensive in bandwidth consumption to help here.
- Passers-by often don't know a programmer is in a dPP session, and will enter an office and begin a conversation; a specific sign must be used to tell this if one does not want a shut door.
- There was a learning curve with dPP that is not present in co-located PP.
- Lack of physical proximity means large amounts of time spent on verbal explanations that could rapidly be resolved by a visual diagrams; although *NetMeeting* has a whiteboard, it is cumbersome to use and does not adequately solve this problem.

6. Conclusions and Future Work

Our experiments support these conclusions about the efficacy of distributed pair programming:

- Pair programming in virtual teams is a feasible way of developing software.
- Our earlier work found that dPP programs were equal in quality to those produced both by co-located pairs and by teams not synchronously paired; these new studies continue to uphold this as well, in that dPP pairs produced *better* programs than dNP teams.
- Effective collaborative software development is possible with a few simple, non-custom, widely-available tools (screen sharing, Internet-based audio communications)
- Feedback from the participants indicates that synchronous pairing (pair programming) engenders better teamwork and communication within a virtual distributed team.
- Distributed pairs maintain many of the benefits (pair pressure, pair learning, two brains) seen in co-located pairs

The studies have some limitations, which we seek to get beyond with further experiments. We are currently studying the following dPP and dXP issues and questions:

Sample size. We plan to repeat these case studies to build up a larger base of results.

Teams vs. pairs. We plan to run larger dPP efforts requiring more than a single pair per team.

Whiteboard, pointing, and facial expressions. As in earlier experiments, we continue to see pairs needing better capabilities for indicating areas of interest ("pointing") and whiteboard use. While *NetMeeting* has a built-in whiteboard, the participants found it limited and awkward to use, and we suspect all software whiteboard programs will be the same. The problem is size, and using wrist muscles to do drawing (not natural). The participants also indicated a desire to see facial expressions, but Webcam's were ineffective for the reasons cited above.

To investigate these problems we are doing follow-on experiments with a video-enhanced dPP environment [16]. The environment uses 2 PCs: one with the screen sharing infrastructure used here, and the other projecting a full screen image of the partner on a wall to the side of the programmer, in arm's reach. We have a whiteboard digitizer on this projection surface. Pair members can easily shift off video, then reach out and draw normally (with virtual ink); the drawings are shared and are projected at the partner's site. A button push restores video.

No "chit chat". We had one programmer make an interesting comment about the technical infrastructure and the fact that it is not as "seamless and glitchless as face-to-face conversation." This participant had developed several programs using co-located pair programming in a class at UNC. He then participated in one of the dPP developments. When asked to compare the experiences, he noted that in co-located pair programming he and his partner has spent a fair amount of time "chit chatting" and that this was not possible (or did not happen to near the same degree) in the dPP infrastructure. This comment could be taken to mean the dPP infrastructure provides a decreased capability for human, team-building interactions; his implication, however, was that the dPP infrastructure oddly enough *increased* productivity by offering slightly *less* fluid interactions. He suggested that the communications mechanisms, while adequate and effective for code development, were not smooth enough to encourage extraneous talking. We find this an interesting point for further investigation.

Acknowledgements We gratefully acknowledge Intel for providing Webcam equipment, Symantec for providing *pcAnywhere* software, and IBM for donating PC equipment in support of our experiments. We would also like to recognize NCSU graduate student Vinay Ramachandran for developing the *Bryce* tool for recording project metrics. Our research was also partially supported by the US Environmental Protection Agency under grant # R82-795901-3.

References

[1] Baheti, P., Gehringer, E., and Stotts, D., "Exploring the Efficacy of Distributed Pair Programming," Proceedings Extreme Programming/Agile Universe, Chicago, IL, 2002.

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

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

[4] Beck, K., *Test Driven Development -- by Example*. Boston: Addison Wesley, 2003.

[5] Cockburn, A. and Williams, L., "The Costs and Benefits of Pair Programming," in *Extreme Programming Examined*, G. Succi and M. Marchesi, Eds. Boston, MA: Addison Wesley, 2001, pp. 223-248.

[6] Dutoit, A. H., Bruegge, Bernd, "Communication Metrics for Software Development," *IEEE Transactions on Software Engineering*, pp. 615-628, 1998.

[7] Foley, S. P., "The Boundless Team: Virtual Teaming," Seminar in Industrial and Engineering Systems, Master of Science in Technology (MST) Graduate Program, Northern Kentucky University MST 660, July 24, 2000.
[8] George, B. and Mansour, Y. M., "A Multidisciplinary Virtual Team," Proceedings Systemics, Cybernetics and Informatics (SCI) 2002, 2002.
[9] George, B. and Williams, L., "An Initial Investigation of Test-Driven Development in Industry," Proceedings ACM Symposium on Applied Computing, Melbourne, FL, 2003.
[10] Gould, D., "Leading Virtual Teams," *Leader Values (Electronic),* http://www.leader-values.com/Guests/Gould.htm, July 9, 2000.
[11] Humphrey, W. S., *A Discipline for Software Engineering*. Reading, Massachusetts: Addison Wesley Longman, Inc, 1995.
[12] McMahon, P. E., "Distributed Development: Insights, Challenges, and Solutions," *CrossTalk*, pp. http://www.stsc.hill.af.mil/CrossTalk/2001/nov/mcmahon.asp, 2001.
[13] McMahon, P. E., *Virtual Project Management: Software Solutions for Today and the Future*. Boca Raton: St. Lucie Press, 2001.
[14] Nosek, J. T., "The Case for Collaborative Programming," in *Communications of the ACM*, vol. March 1998, 1998, pp. 105-108.
[15] Olson, G. M. and Olson, J. S., "Distance Matters," Proceedings Human Computer Interaction, 2000.
[16] Stotts, D., Smith, J., and Williams, L. A., "A Video-Enhanced Environment for Distributed Extreme Programming," Department of Computer Science. Univ. of North Carolina at Chapel Hill, Chapel Hill, NC TR-02-009, March 1, 2002.
[17] Traub, J., "This Campus is Being Simulated," in *The New York Times Magazine*, 2000, pp. 88-93+.
[18] Williams, L. and Kessler, R., *Pair Programming Illuminated*. Reading, Massachusetts: Addison Wesley, 2003.
[19] Williams, L., Kessler, R., Cunningham, W., and Jeffries, R., "Strengthening the Case for Pair-Programming," in *IEEE Software*, vol. 17, 2000, pp. 19-25.
[20] Williams, L. A., "The Collaborative Software Process PhD Dissertation," in *Department of Computer Science*. Salt Lake City, UT: University of Utah, 2000.