Distributed Pair Programming: An Empirical Study

Brian F. Hanks

School of Engineering
University of California, Santa Cruz
brianh@soe.ucsc.edu

Abstract. Pair programming provides many benefits, both to the programmers and to the product that they develop. However, pair programming is limited to those situations in which the developers can collocate, preventing its benefits from being enjoyed by the widest possible audience. A software tool that allowed the pair to work from separate locations would address this limitation. This paper presents some initial results from a distributed pair programming experiment in which students in an introductory programming class used such a tool. Student perceptions of distributed pair programming are also discussed.

1 Introduction

Pair programming [1] transforms what has traditionally been a solitary activity into a cooperative effort. In pair programming, two software developers share a single computer monitor and keyboard. One of the developers, called the *driver*, controls the computer keyboard and mouse. The driver is responsible for entering software design, source code, and test cases. The second developer, called the *navigator*, examines the driver's work, offering advice, suggesting corrections, and assisting with design decisions. The developers switch roles at regular intervals. Although role switching is an informal process, a typical interval is 20 minutes.

Initial studies suggest that two programmers working together in the pair programming style produce code with fewer errors, with only a slight increase in total programmer time [2]. In addition, programmers who pair report improved mentoring [2], increased confidence in their solutions [3, 4], and increased job satisfaction [2, 5].

At the university level, pair programming has been used in computer science courses as a teaching tool [3, 4]. McDowell et al. [4] found that allowing students to pair in an introductory programming course resulted in a greater percentage of students completing the course. These students also were more likely than non-pairing students to select computer science related majors within one year of completing the course. Compared with students who work alone, students who pair are more likely to turn in solutions to their programming assignments, and these solutions are of higher quality [6, 7].

One major drawback of pair programming is that both members of the pair must be collocated. Although research has shown that collocated work has significant advantages over work by distributed teams [8], there are many factors motivating distributed work. More workers are telecommuting, and many organizations have offices in multiple locations, resulting in geographically dispersed project teams.

This trend towards distributed teams conflicts with the collocation requirement of pair programming. By removing the collocation requirement, the benefits of pair programming could by enjoyed by a wider audience. A tool that supports *distributed pair programming*, in which the driver and navigator pair from separate locations, would remove this impediment to the adoption of pair programming. I have developed such a tool, and have conducted a controlled experiment to evaluate its effectiveness.

2 Related Work

Schumer and Schumer [9] and Maurer [10] have conducted research in this area that suggests that distributed pair programming (DPP) can work. However, their research was more focused on distributed XP, and provides only anecdotal evidence that DPP is effective. No quantitative results are provided and no comparison is made between distributed and collocated pair programming.

More recent work by Baheti et al. [11] suggests that distributed pairing can be as effective as collocated pairing. Students who virtually paired using Microsoft's NetMeeting performed similarly to collocated pairs in terms of the grades they received on their project. Because student pairs worked on separate, self-selected projects direct comparison of their performance is difficult.

Canfora et al. [12] studied virtual pairing by having students use a screen sharing application along with a text-based chat application. No audio channel was provided to the students. They found that distributed pairs tended to stop cooperating and began to work as two solo programmers. Based on these results, they made some suggestions for features that should be supported in a distributed pair programming tool. However, these suggestions may not be valid because the distributed pairs in this study did not have a method of speaking with each other.

Stotts et al. [13] provides further evidence of the potential success of distributed pairing. They describe an on-going series of experiments and case studies in which students virtually paired. Although distributed pairs successfully completed their programming assignments, they complained of their inability to point or gesture. As Stotts observed, "pairs need better capabilities for indicating areas of interest".

While this research provides compelling evidence that DPP can be effective, it seems clear that a tool that supported the needs of the distributed pair would enhance DPP. The next sections describe such a tool, an experiment that I conducted to evaluate its effectiveness, and some initial experimental results and user experiences.

3 VNC for Pair Programming

The tool described here is based on the open source screen sharing application Virtual Network Computing (VNC) [14]. VNC allows a user’s desktop to be replicated onto multiple computers (in particular, two in the case of pair programming). Application output is displayed on both computers, while keyboard and mouse input from either computer is sent to the applications.

I decided to use a screen sharing approach in the development of this tool, instead of developing collaboration-aware distributed pair programming development tools. This choice was motivated by the fact that screen sharing applications allow single user applications to be shared by multiple users without modification [15]. I believe that this is especially important for complex tasks such as software development. Software developers use a large set of tools in their work, such as compilers, editors, debuggers, and testing tools. It is critical for a distributed pair of developers to be able to use their preferred development tools, and the screen sharing approach supports this requirement.

While VNC can be used as-is for DPP, it is not ideal for this. Both users have active keyboards and mice. If both partners use the keyboard simultaneously, their keystrokes are interlaced into an unintelligible stream. There is also only one cursor, which makes it difficult for the navigator to point at areas of the screen. [13]

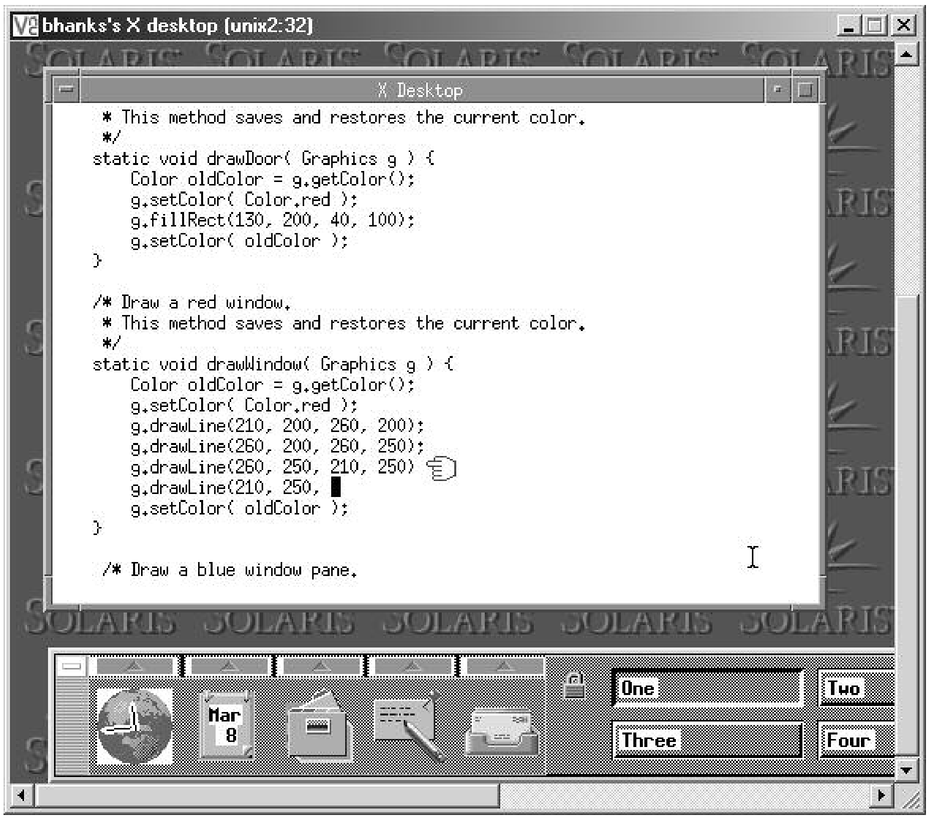
I have modified VNC to provide a second cursor that can be controlled by the navigator. This cursor is displayed when the navigator presses the F8 key to enter 'gesture mode'. In gesture mode, a second cursor (in the form of a red hand with a pointing index finger) is added to the user interface. The navigator's mouse controls the movements of this cursor, while the driver retains control of the standard cursor. While the navigator is in gesture mode, keyboard and mouse button events are ignored, and mouse movement events are used to control the position of the gesturing cursor.

This second cursor allows the navigator to point at areas of the screen without affecting the driver's state. Figure 1 shows a portion of the driver's desktop while the navigator is gesturing. In this figure, the driver is typing in an editor window while the navigator is pointing at the previous line where there is a missing semicolon.

While many synchronous groupware applications provide a separate cursor for each user [16], my approach differs from the typical one in that the cursor is transient and only present when the navigator requests it. A benefit of this approach is that the navigator's intent is clear to the driver, because the second cursor only appears when the navigator wants to gesture. A continuously present second cursor would not be as noticeable to the driver, and gesturing might not be as readily observed.

The shape of the cursor also plays a role in its usability [17]. In an earlier version of this tool the gesturing cursor was arrow-shaped. It was changed to its current hand shape based on early user feedback.

In addition to frequent gesturing, collocated pair programmers converse regularly during a pairing session [1]. It is essential for partners in a distributed pair to be able to speak with each other. One obvious solution is for the partners to use telephones. However, in the experiment described in this paper, the students who participated did not have always have access to telephones. Instead, they used a voice-over-IP application (such as AOL Instant Messenger) and computer headsets to converse. This also allowed them to use the chat window of AIM to transmit items that are difficult to convey verbally, such as URLs, long path names, and complex method names.



```
 * This method saves and restores the current color.
 */
static void drawDoor( Graphics g ) {
    Color oldColor = g.getColor();
    g.setColor( Color.red );
    g.fillRect(130, 200, 40, 100);
    g.setColor( oldColor );
}

/* Draw a red window.
 * This method saves and restores the current color.
 */
static void drawWindow( Graphics g ) {
    Color oldColor = g.getColor();
    g.setColor( Color.red );
    g.drawLine(210, 200, 260, 200);
    g.drawLine(260, 200, 260, 250);
    g.drawLine(260, 250, 210, 250)
    g.drawLine(210, 250,
    g.setColor( oldColor );
}

 /* Draw a blue window pane.
```

Figure 1: Gesturing Mode

4 The Experiment

In the fall 2003 quarter, 76 students in two sections of an introductory programming course at the Santa Cruz campus of the University of California volunteered to participate in an experiment to evaluate distributed pair programming. All students in these sections pair programmed, even if they were not experimental volunteers. Students were not paid to participate in the experiment.

Potential volunteers were told that I was studying a new tool to support pair programming, but they were not given any details regarding the tool. In particular, they were not told that the tool would allow them to work from separate locations.

Separate instructors taught these two classes. Although the courses covered the same material, the programming assignments and exams were different. The number of programming assignments also differed between the two sections – one section had five graded assignments while the other had seven. The author was the instructor of the section with seven assignments, which is identified as section 2 in the discussion that follows.

Volunteer pairs were randomly assigned to one of two groups. One group was allowed to pair program using the tool described here, while the second group paired while collocated. The pairs in the tool group were allowed to use the tool as much or as little as they wanted. They could still pair while collocated if it was convenient for them, such as immediately before or after class.

Although allowing students in the tool group to pair while collocated was not ideal from an experimental viewpoint, it is important to remember that the experiment was being conducted in a classroom setting. Because of this, it would not have been ethical to force the students in one group to act in a way that might have negatively impacted their classroom performance.

Students in both groups initially worked on their programming assignments while collocated, so that they could establish good working relationships with their partners before attempting distributed pairing. One reason for this is the pair jelling factor noted by Williams [3], which indicates that students have to learn to pair. Student pairs became more effective in comparison to individuals after they had completed their first assignment. A second reason for having students work together before attempting distributed pairing is that I believe that they are more likely to have established a cooperative relationship. This is substantiated by Rocco’s [18] findings that individuals who had prior face-to-face interaction were more likely to cooperate in a distributed game than those who had never met. Stotts et al. [13] also noted that face-to-face meetings helped distributed pairs work effectively.

After this initial period of collocated work, the pairs in the tool group were instructed in its use. Computer headsets were loaned to these students. These students were allowed to pair while using the tool on the remainder of their programming assignments. The students in the control group continued to pair while collocated.

Student volunteers filled out surveys at the beginning and end of the experiment. These surveys contained questions about the students' demographic data, experience, and opinions about pair programming. The students in the tool group were asked additional questions about their experience using the tool.

The volunteers also answered a few questions (using a form on a web site) when they turned in their programming assignments. Each student was asked how much time they spent driving, navigating, and working alone, how confident they were in their solution to the assignment, and how satisfied they were with their working relationship with their partner on the assignment.

5 Experimental Results

To be successful in an academic setting, a tool must not have a negative impact on student performance. One measure of student performance is the final exam. Another measure is the students' confidence, as this directly influences their success in computer science courses.

It is also important to measure user satisfaction with the distributed pairing process. Subjects were asked to rate the utility of the gesturing feature, and comment on what they liked and disliked about distributed pairing.

5.1 Final Exam Performance

Of the 76 students who volunteered for the experiment, 72 took the final exam. This was an individual, in-class, written exam with a variety of question types. Students were expected to understand and create short programs, methods, or classes. Table 1 shows the final exam scores for these students. Because the students in the two sections did not take the same final exam, the results for the two sections cannot be compared directly. The categories in the table are (1) All: all student volunteers as a group, (2) Non Tool Group: the collocated students, (3) Tool Group: the students who were allowed to use the tool, and (4) Tool Users: those students in the tool group who actually used the tool. As shown in the table, students in the collocated and distributed groups performed equally on the final exam. Distributed pairing did not negatively affect student performance on the final exam.

	Group	**Final Exam Score**	**Number of Subjects**
Section 1	All	79.0	31
	Tool Group	78.6	15
	Non Tool Group	79.3	16
	Tool Users	83.0	4
Section 2	All	69.7	41
	Tool Group	69.8	21
	Non Tool Group	69.7	20
	Tool Users	73.6	14

Table 1: Final Exam Score

Curiously, those students in the Tool Group who actually used the tool on their homework assignments performed better on the final exam, although this difference is not statistically significant. A possible explanation for this difference is that higher performing students are more likely to make the effort to try a new software tool.

5.2 Confidence

When they turned in their programming assignments, each student was asked to respond to the following question: "On a scale from 0 (not at all confident) to 100 (very confident), how confident are you in your solution to this assignment?”. Table 2 shows the mean values of the responses and associated statistical significance for the students in the two sections of the class.

In section 1, students in the tool group began using the tool on their third programming assignment. In section 2, they began using the tool on the fourth assig ment. There were no statistically significant differences in student confidence in either section on any of the programming assignments, either before or after the students began using the tool.

	Assignment	Non Tool Group	Tool Group	ANOVA	p
Section 1	1	94.57	95.73	F(1,27) = 0.28	0.60
	2	97.31	85.13	F(1,26) = 2.75	0.11
	3	96.92	92.81	F(1,27) = 2.10	0.16
	4	70.11	84.40	F(1,22) = 1.19	0.29
	5	45.45	67.28	F(1,23) = 1.73	0.20
Section 2	1	97.79	95.56	F(1,31) = 2.27	0.14
	2	83.58	84.63	F(1,31) = .015	0.91
	3	89.61	94.11	F(1,34) = 0.74	0.41
	4	92.88	91.12	F(1,32) = 0.34	0.56
	5	91.78	89.37	F(1,32) = 0.31	0.58
	6	73.80	58.36	F(1,27) = 1.44	0.24
	7	90.88	82.44	F(1,31) = 2.34	0.14

Table 2: Student Confidence

5.3 Gesturing

To evaluate the effectiveness of the gesturing feature in enabling DPP, the students who used the tool were asked to indicate their level of agreement with the statement, "The gesturing feature was very useful to me." The response was given on a 7 point Likert scale, where 1 meant “strongly disagree”, 4 meant “neutral”, and 7 meant “strongly agree”.

Figure 2 shows the students' responses. The mean response was 5.47. Although one student disagreed with the statement and two were neutral, the other 14 were in agreement. These results are encouraging, and indicate that the gesturing feature aids the distributed pairing process.

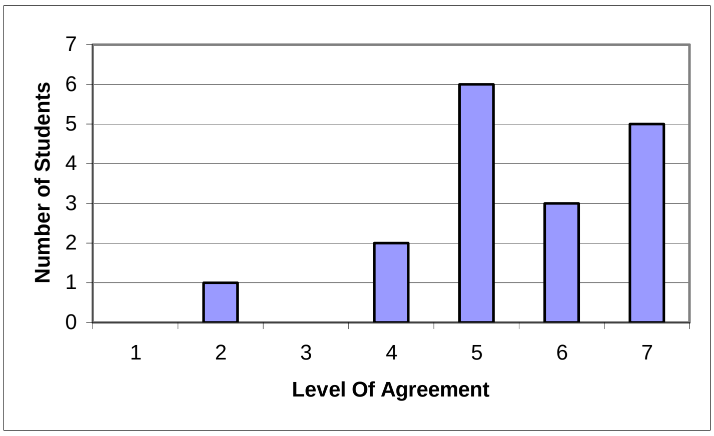


Figure 2: Gesturing Feature was Useful to Me

6 Student Experiences with the Tool

To gain more understanding of the benefits and drawbacks of DPP, students were asked what they liked and disliked about using the tool. I was also interested in learning why some students in the tool group chose not to use it.

6.1 Student Likes

Students gave a variety of answers to the question, "What did you like about the distributed pair programming tool?". Many of the responses addressed convenience and effective use of time. A representative sample of these responses includes:

- "Well, besides it allowing us to work in the comforts of our own homes without ever getting out of our chairs, it also helped to overcome some schedule conflicts, and the time that would have been wasted just walking to the other person's computer was instead turned into productive programming time!"
- "You don't have to go all the way to a computer lab to pair program."
- "It made pair programming very easy and convenient. We didn't have to meet on campus or at each other's houses so we could always pair program w/out the effort of getting together. I think the class would have required a lot more time w/out the tool."

Another set of responses addressed student satisfaction with the tool itself. These responses indicated that the students felt that the tool was an effective way for them to pair program without having to physically meet. Responses of this type include:

- "If we didn't meet in person, this combined with AIM almost perfectly emulated working side by side. We could work on it any time and take long breaks."
- "The pair programming tool allowed us to work together from two different places. The pointing function of the program also made it easier to point out errors and not accidentally type while my partner was typing."

An interesting group of responses came from students who used the tool *while collocated*. These students found that using the tool improved the pairing process even when they weren't separated. In fact, some of these students never used the tool from separate locations, but still found it valuable. Comments from this group of students includes:

- "I like the flexibility it offers in case two partners can't meet and work together. I liked being able to work on separate terminals while working side by side in the lab."
- "Easier than sharing computer"
- "Being able to switch driver/navigator easily"

These students felt that the typical physical setup in the campus computing labs made role switching difficult. These labs are set up for solo work, and it is sometimes difficult to get two chairs in front of one workstation. The limited desk space also makes moving the keyboard and mouse difficult, and hinders the role switching process. The tool eliminated these problems, and allowed the students to use any campus computing lab as if it were a pair programming lab.

One particularly interesting comment was made by a female student, who said, "The tool was useful at night when we couldn't work in the lab together because we live on opposite sides of campus." This comment suggests that some students may feel uncomfortable on campus at night, and that tools such as this would allow them to work in a safe environment.

6.2 Student Dislikes

Students were also asked what they disliked about the distributed pairing tool. Many of the comments had to do with problems with the computing environment, such as poor quality audio, or the lack of a Macintosh version of the tool.

Some students seemed to feel disconnected from their partner while distributed. Typical comments from these students include:

- "Communication with the partner is still awkward"
- "The tool was difficult to use when we were programming something we had never programmed before – for instance, when we first used arrays."
- "We sometimes wrote over each other's work and sometimes presenting things in person kept each others' interest."
- "AIM is not a good way to communicate, even with the headsets. Sometimes it is difficult to explain something through the air."

One particularly interesting comment came from a student who felt that distributed pairing was antisocial:

- "Discourages human contact. One of the things I like best about pair programming is that the two people are physically in the same place together, talking and interacting face-to-face. This helps foster community and social interaction. The tool makes face-to-face contact unnecessary, and this can encourage a retreat into the "lone-wolf" mode, an aspect that troubles me"

6.3 Reasons for Not Using the Tool

The students in the tool group did not have to use it, and some of them chose not to. I asked these students the question, "Why didn't you use the distributed pairing tool?".

One common reason for not using the tool was that they didn't find it necessary. Many of the pairs found it easy to physically meet with their partner, and therefore did not see any reason to use the tool. Comments from these students included:

- "We never had any need to. It was easier to meet in person."
- "Because we were able to find time together working on it at one person's place"
- "It was not necessary for us. It was very easy for us to meet in lab and talk face to face"

Some students found it challenging to get the program running. They had to start up a server application, connect with the partner using AIM, and then run the VNC client. Comments from these students include:

- "Too many programs to do a simple thing."
- "It was a hassle trying to get the program up and running than just simply meeting up with your partner at the lab. "

It is clear that some students don't see any need for distributed pairing. It was easy for them to meet with their partner, and they were happy to do so.

The tool may have helped some of the other students in the tool group who did not use it. These students gave up in their attempts to use the tool because it was not easy enough for them to set up. It is important to remember that these students are not necessarily experienced computer users, and the steps required to establish a distributed pair programming session may have been too complex. This suggests that more work needs to be done to make the tool simpler to setup.

7 Concluding Remarks

Although previous research has shown that DPP is possible, none of this research investigated tools specifically designed to support distributed pairing. This paper describes a tool that was developed specifically to support distributed pairing, and reports results showing that students who used the tool performed as well as collocated students, had similar levels of confidence, and found the tool beneficial.

Students in the control and experimental groups performed equally well on the final exam. Although it is not statistically significant, students who used the tool performed better on the exam than the students in the control group. Students in both experimental groups were also equally confident in their programming solutions.

Students agreed that the gesturing feature was useful to them. Further research is needed to verify the usefulness of the gesturing feature by comparing the performance of pairs using this tool with that of pairs using VNC without the gesturing feature.

This paper discusses student performance in terms of final exam scores. This is not the best measure of pair programming performance, for a couple of reasons. First, the final exam is an individual effort. Second, it measures student understanding of the course material, but does not directly measure programming ability. The students' grades on their programming assignments would provide a better indication of the impact of distributed pair programming on student performance. This analysis remains to be done.

The results reported here may not generalize to other populations. The experimental subjects were students who were learning to program; therefore, these results may not be applicable to DPP with experienced software developers. Similarly, the experimental subjects were allowed to establish working relationships with their partners before using the tool, and those in the tool group were also allowed to pair while collocated. The results reported here may not apply to those situations where partners are not able to physically meet before or while pairing.

As noted earlier, there are many situations where collocated pair programming cannot be done. The availability of an effective DPP tool would allow the benefits of pair programming to be enjoyed by a larger audience. Although much work remains to be done to develop a commercial grade tool, the results presented here show that such a tool can facilitate distributed pairing.

Acknowledgements

This research is supported by National Science Foundation grant DUE-0341276. Plantronics, Inc. also supported this research by generously donating computer headsets. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the author and do not necessarily reflect the views of the National Science Foundation or those of Plantronics.

References

1. Laurie Williams and Robert Kessler. *Pair Programming Illuminated*. Addison-Wesley, 2002.
2. Alistair Cockburn and Laurie Williams. The costs and benefits of pair programming. In Giancarlo Succi and Michele Marchesi, editors, *Extreme Programming Examined*, pages 223 - 247. Addison-Wesley, 2001.
3. Laurie A. Williams. Strengthening the case for pair programming. *IEEE Software*, 17(4):19 - 25, July/August 2000.
4. Charlie McDowell, Linda Werner, Heather Bullock, and Julian Fernald. The impact of pair programming on student performance, perception and persistence. In *Proceedings of the International Conference on Software Engineering (ICSE 2003)*, pages 602 - 607, May 3 - 10, 2003.
5. John T. Nosek. The case for collaborative programming. *Communications of the ACM*, 41(3):105-108, March 1998.
6. Charlie McDowell, Brian Hanks, and LindaWerner. Experimenting with pair programming in the classroom. In *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education*, 2003.
7. Brian Hanks and Charlie McDowell. Program quality with pair programming in CS1. To appear in *Proceedings of the ninth annual conference on innovation and technology in computer science education (ITiCSE*), June 28 - 30, 2004.
8. Judith Olson, Stephanie Teasley, Lisa Covi, and Gary Olson. The (currently) unique advantages of collocated work. In Pamela Hinds and Sara Kiesler, editors, *Distributed Work,* pages 113-135. The MIT Press, 2002.

9. Till Schummer and Jan Schummer. Support for distributed teams in extreme programming. In Giancarlo Succi and Michele Marchesi, editors, *Extreme Pro gramming Examined*, pages 355-378. Addison-Wesley, 2001.
10. Frank Maurer. Supporting distributed extreme programming. In *Extreme Programming and Agile Methods - XP/Agile Universe 2002,* number 2418 in LNCS, pages 13-22. Springer, 2002.
11. Prashant Baheti, Edward Gehringer, and David Stotts. Exploring the efficacy of distributed pair programming. In *Extreme Programming and Agile Methods - XP/Agile Universe 2002*, number 2418 in LNCS, pages 208-220. Springer, 2002.
12. Gerardo Canfora, Aniello Cimitile, and Corrado Aaron Visaggio. Lessons learned about distributed pair programming: What are the knowledge needs to address? In *Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE03),* pages 314-319, 2003.
13. David Stotts, Laurie Williams, Nachiappan Nagappan, Preshant Baheti, Dennis Jen, and Anne Jackson. Virtual teaming: Experiments and experiences with distributed pair programming. In *Extreme Programming and Agile Methods - XP/Agile Universe 2003,* number 2753 in LNCS, pages 129-141. Springer, 2003.
14. Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33-38, January-February 1998.
15. Saul Greenberg. Sharing views and interactions with single-user applications. *ACM SIGOIS Bulletin*, 11(2-3):227-237, April 1990.
16. Stephen Hayne, Mark Pendergast, and Saul Greenberg. Gesturing through cursors: Implementing multiple pointers in group support systems. In *Proceedings of the 26th Hawaii International Conference on System Science*, volume 4, pages 4-12, 1993
17. Saul Greenberg, Carl Gutwin, and Mark Roseman. Semantic telepointers for groupware. In *Proceedings of the 6th Australian Conference on Computer-Human Interaction*, pages 54-61, 1996.
18. Elena Rocco. Trust breaks down in electronic contexts but can be repaired by some initial face-to-face contact. In *Proceedings of CHI 98*, pages 496-502, 1998.