

Exploring the efficacy of distributed pair programming

Prashant Baheti

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

Dr Edward Gehringer

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

Dr David Stotts

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

Abstract.

Pair programming is one of the twelve practices of Extreme Programming (XP) [1]. Pair programming is usually performed by programmers who are *collocated*—working in front of the same monitor. But the inevitability of distributed development of software gives rise to important questions: 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 different working arrangements of student teams developing object-oriented software. Teams were both collocated and in distributed environments; some teams practiced pair programming while others did not. In particular, we compared the software developed by virtual teams using distributed pair programming against collocated teams using pair programming and against virtual teams that did not employ distributed pair programming. 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 (without pair programming) in terms of productivity and quality.

Keywords:

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

1. Introduction

Increasingly, programmers are working in geographically distributed teams. The trends toward teleworking, distance education, and globally distributed organizations are making these distributed teams an absolute necessity. These trends are beneficial in many ways, particularly for those in geographically disadvantaged areas. However, it is not believed that any of these arrangements makes programming more effective than if all the programmers were, indeed, collocated. Therefore, organizations must strive to maximize the efficiency and effectiveness of these unavoidably distributed programmers and teams. This paper describes the development and study of a technique tailored for distributed programming teams.

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 typing at the computer or writing 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 [4, 6].

Pair programming is one of the twelve practices of Extreme Programming (XP) [1]. It is usually assumed that the pairs will be working in front of the same workstation. If Extreme Programming is to be used for distributed development of software, collocation becomes a limitation. Indeed, it would require a variant of Extreme Programming in which distributed pair programming or virtual teaming can be used.

By *distributed pair programming*, we mean that two members of the team (which may consist of only two people) synchronously collaborate on the same design or code, but 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 or even capability of video conferencing with Web cams if required. This paper is based on an experiment where these students developing distributed team projects had this capability and how they fared against those that developed software in a collocated manner or in distributed teams but without pair programming.

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 hypotheses based on which we test our results. Section 4 describes an initial experiment was done in fall 2001 between NCSU and UNC to determine an effective technical platform to allow distributed pair programming the hypothesis for which we test our results. Section 5 outlines the comprehensive experiment that was conducted in a graduate class at NC State University. Section 6 presents the results, followed by an outline of future work in Section 7. The conclusions are presented in Section 8.

2. Previous Work

2.1 Pair programming

Previous research [4, 5] has indicated that pair programming is better than individual programming in a collocated environment. 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 [4, 6,7,8].

Do these results also apply to distributed pairs? It has been established that distance matters [3]; face-to-face pair programmers will most likely outperform distributed pair programmers in terms of sheer productivity. But the inevitability of distributed work has made it important to discover the most effective way to develop software in a distributed environment. The traditional way to develop distributed software is with virtual teams, where the collaboration between the team members is asynchronous, as opposed to using distributed pair programming.

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 [2]. 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 [9].

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" [11].

In comparison to collocated teams, virtual teams have three disadvantages: "Communication within the team is hindered, team members are less aware of each other and common access to physical objects and places (like a printer or the cafeteria) is difficult" [10]. The first two of these can be addressed using the techniques of Computer Supported Cooperative Work (CSCW). As for the third one, physical objects can be simulated if they have must have meaningful electronic representations like virtual rooms or electronic blackboards.

A primary consideration in virtual teaming is that of communication [12]. Poor communication can cause problems like inadequate project visibility, wherein everyone works individually, but no one knows if the pieces can

be integrated into a complete solution. The communication problem is alleviated by the use of groupware applications like e-mail systems, messaging systems and videoconferencing.

According to Dourish and Bellotti [13], "Awareness is an understanding of the activities of others, which provides a context for your own activity. This context is used to ensure that individual contributions are relevant to the group's activity as a whole, and to evaluate actions with respect to group goals and progress". Software like TUKAN [10] provide a synchronous distributed team programming environment to deal with the awareness issues.

3. Hypotheses

In the fall of 2001, we ran an 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. In the academic backdrop, quality can be assessed by the grades obtained by the students for their project. A statistical *t*-test can be performed to find whether one of the groups gets statistically significantly better results at different levels of significance ($p < 0.01, 0.05, 0.1$ etc.).
- 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. Initial Platform Experiment

An initial experiment was conducted in early fall 2001 between NCSU and UNC to determine an effective technical platform to allow distributed pair programming. Two pairs of programmers worked as a 4-person team over the Internet to develop a modest Java gaming application; each pair was composed of one programmer from each remote site. The team developed a Mancala game, with GUI, in 8 sessions that varied from 1 to 2 hours in length. In addition to the actual pair-programming sessions, the project was initiated with a face-to-face meeting in which the team members agreed on requirements and an overall system metaphor. Thus the experiment mainly tested the effectiveness of the technology for *pair coding* and not the entire software development process.

The members of a pair viewed a common PC display using desktop sharing software; we tried Microsoft NetMeeting, Symantec's PCAnywhere, and VNC. They used headsets and microphones to speak to each other, and text chat for communications as well. We tried several instant-messaging programs (Yahoo Messenger, PalTalk, AOL Messenger) before implementing the project. The final experiment was run with NetMeeting, as this program provided PC sharing, text, audio, and video in one platform.

A typical pairing 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. The students were furnished Intel digital cameras to use as Web cams for videoconferencing, to allow them, for example, to show paper design documents to each other. However, as the sessions progressed, none of these teams found the need to use the Web cams. One of the students described his experience as follows:

"While we programmed, I kept implementing the methods described in the class diagram and he kept guiding me wherever he felt I was straying in logic or when I made typing errors. His style of programming was a bit different than mine. I have done more of ad hoc programming where I didn't need

to make the code readable. But for this project, I realized that we would have to put proper names for the variables so that by seeing the code itself, one could understand its function. So, that changed my way of coding and I got more organized.”

“I did not meet with any difficulties while programming distributedly, as he was constantly giving me input. Whenever we got stuck at a point, we discussed how to get over the problem, either by modifying the class diagrams or introducing new methods for functionality that we had missed out on.”

“All in all, once the technology falls into place, its not difficult to program in pairs in a distributed environment. In fact, things go faster than usual because two people are thinking on the same problem. Also, you try to program better because you know that someone is supervising your work. It has been a unique and pleasant experience.”

Our goal was not to test whether a remote pair could be as efficient as a co-located one, but simply to see if the programming pairs could work well enough to make functional software in reasonable time. The pairs reported that after a few early sessions in which they were learning the platform behavior, they felt comfortable and natural coding with this technology. The final game works correctly. From this experiment we found that effective remote teaming could be done with the PC sharing software and audio support. This platform was then used in the more comprehensive controlled experiment described next.

5. The Main Experiment

The experiment was conducted in a graduate class, Object-Oriented Languages and Systems,¹ taught by Dr Edward Gehringer at North Carolina State University. The course introduces students to object technology and covers OOA/OOD, Smalltalk, and Java. At the end of the semester, all students participate in a 5-week long team project. The team projects consisted of programming projects like updating a GUI to use JSP, implementing a dynamic reviewer-mapping algorithm for peer review, simulating the LC-2 architecture, or building a GUI for DC motor control. Some of the projects were specified by the instructor, and some were specified by a sponsor on the NCSU ECE faculty. 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 videotapes² 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.

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.

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.

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

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

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.

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.

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 platform experiment was done in early fall 2001 between NCSU and UNC helped in determining an effective technical platform to allow remote teaming.

The pairs in our full controlled 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 Web cams 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 Web cams.

In order to record their progress, the students utilized an online tool called Bryce [14], a Web-based software-process analysis system used to record metrics for software development. Bryce was developed at N.C. State. 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 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.

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

6.1 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 Fig. 1.

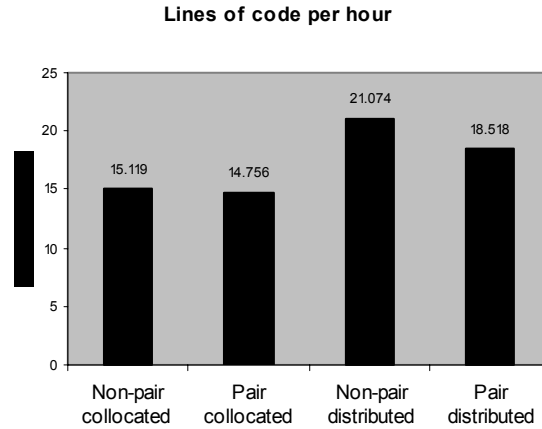


Fig. 1.

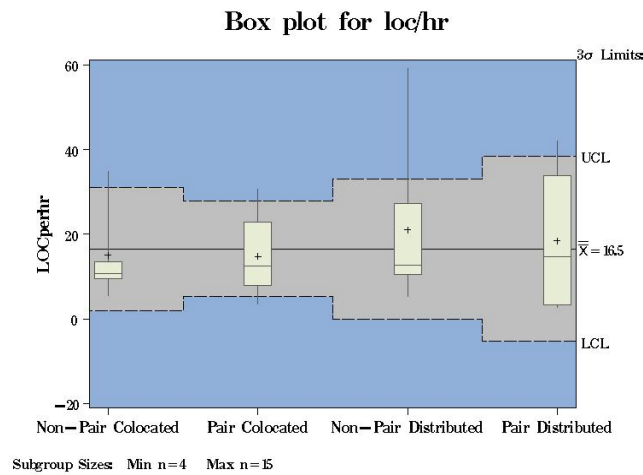


Fig. 2.

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 (Fig. 2) for the four categories, which illustrates the distribution of the metric for the four environments.

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 are not shown to be worse in terms of productivity than those forming virtual teams without distributed pair programming.

To find if the collocated pairs fared any better than distributed pairs, a *t*-test was also conducted for these two categories, and again, no category was found to be statistically significantly better than the other. Hence, it can be concluded that collocated pairs for this experiment were not more productive (statistically) than distributed pairs.

6.2 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. We consider grade to be a good metric of quality because the grades were given after half-hour long demos to the TA assigned to a particular category of projects, sometimes in the presence of the project sponsor. The graph (Fig. 3) below indicates that the performance of students did not vary much from one category to another.

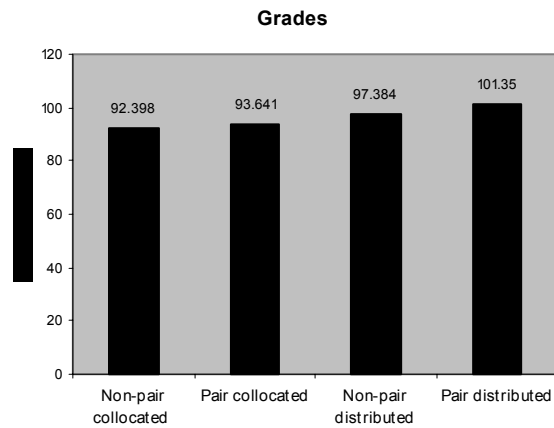


Fig. 3.

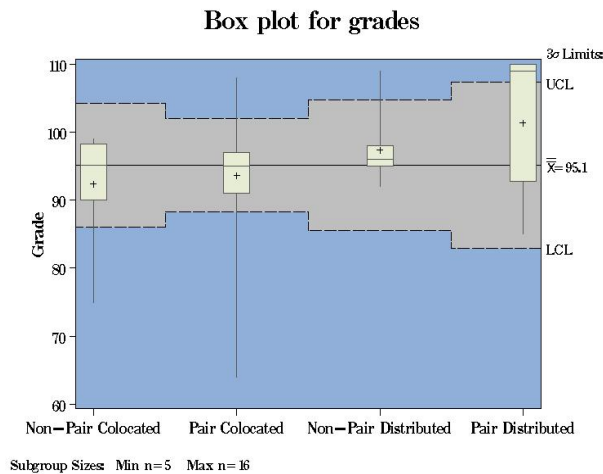


Fig. 4.

A box plot (Fig. 4) 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 virtual teaming were not significantly better than the distributed teams using pair programming, in terms of grade.

As in the previous section, a statistical *t*-test was performed between distributed pairs and distributed non-pairs and between collocated pairs and distributed pairs. In either case, the results for difference between the two groups under comparison were not found to be statistically significant. This indicates that distributed pairs are comparable to collocated pairs and distributed non-pairs in terms of quality.

6.3 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 within your team members"

Table 1: Cooperation within the team

Communication among 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. 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 six students involved in distributed pair programming thought that technology was not much of a hindrance in collaborative programming. Also, about the same fraction (82%) of students involved in virtual teaming with or without pair programming felt that there was proper cooperation among team members.

7. 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.

8. Conclusions

The results of our experiment indicate the following:

- Distributed pair programming in virtual teams is a feasible way of developing object-oriented software.
- The results of the experiment indicate that software development involving distributed pair programming is comparable to that developed using collocated pair programming or virtual teams without distributed pair programming. The two metrics used for this comparison were 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.

9. 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.

10. References

- [1] K. Beck, “Extreme Programming Explained: Embrace Change”. Reading, Massachusetts: Addison-Wesley, 2000.
- [2] B. George., Y. M. Mansour, “A Multidisciplinary Virtual Team”, Accepted at *Systemics, Cybernetics and Informatics (SCI)*, 2002.
- [3] G. M. Olson and J. S. Olson, “Distance Matters”. *Human-Computer Interaction*, 2000, volume 15, p. 139–179.
- [4] L. A. Williams, “The Collaborative Software Process PhD Dissertation”, Department of Computer Science, University of Utah. Salt Lake City, 2000.

- [5] J. T. Nosek, “The case for collaborative programming”, *Communications of the ACM* 41:3, March 1998, p. 105–108.
- [6] L. A. Williams, and R. Kessler, *Pair Programming Illuminated*, Boston, MA: Addison Wesley, 2002.
- [7] 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.
- [8] 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
- [9] 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.
- [10] T. Schummer and J. Schummer, “Support for Distributed Teams in Extreme Programming”, in *Extreme Programming Examined*, Succi, G., Marchesi, M. eds., p. 355–377, Boston, MA: Addison Wesley, 2001
- [11] 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.
- [12] D. Gould, “Leading Virtual Teams”, Leader Values, <http://www.leader-values.com/Guests/Gould.htm>. July 9, 2000.
- [13] P. Dourish, V. Bellotti. “Awareness and Coordination in Shared Workspaces”, CSCW '92, *Conference Proceedings on Computer-Supported Cooperative Work*, 1992.
- [14] <http://bryce.csc.ncsu.edu/tool/default.jsp>

11. Biography

Prashant Baheti is a student at North Carolina State University since fall 2001 and is pursuing his Masters with thesis option in Computer Science.

Dr Edward F. Gehringer has been on the faculty of North Carolina State University since 1984, and has taught over 1000 students since 1987. He has published approximately two dozen refereed papers on various aspects of object technology, and has been a frequent contributor to OOPSLA Educators' Symposia.

David Stotts is an associate professor in the Department of Computer Science at the University of North Carolina at Chapel Hill and currently is serving as associate chair for Academic Affairs. Dr. Stotts's research interests include formal methods in software engineering, concurrent computation models, hypermedia, and collaborative distributed systems. From 1990 through 1995, Dr. Stotts served as an ACM distinguished lecturer. He served as general chair for the 1996 ACM Hypertext Conference, and is on the editorial boards of the *Journal of Digital Information* and *World Wide Web*.