

Technical Report TR02-010

Department of Computer Science
Univ of North Carolina at Chapel Hill

Distributed Pair Programming: Empirical Studies and Supporting Environments

**Prashant Baheti, Laurie Williams,
Edward Gehringer**

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

David Stotts, Jason McC. Smith

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

stotts@cs.unc.edu

March 15, 2002

Distributed Pair Programming: Empirical Studies and Supporting Environments

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

David Stotts, Jason McC. Smith
Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599-3175
+1 919-962-1833
stotts@cs.unc.edu

ABSTRACT

Previous research [1, 2] has indicated that pair programming is better than individual programming when the pairs are physically colocated. However, important questions arise: How effective is pair programming if the pairs are not physically next to each other? What if the programmers are geographically distributed? An experiment was conducted to compare the different working arrangements of student teams developing object-oriented software. The teams were both colocated and in distributed environments; some teams practiced pair programming while others did not. The results of the experiment indicate that it is feasible to develop software using distributed pair programming, and that the resulting software is comparable to software developed in colocated or virtual teams. Our early experiments have led to the creation of a more comprehensive environment for support of distributed pair programming, using dual screen projectors and hypermedia-enhanced video streams. Our findings will be of significant help to educators dealing with team projects for distance-learning students, as well as organizations that are involved in distributed development of software.

Keywords

Extreme Programming, XP, pair programming, collaborative software engineering, distance education, virtual team, video hyperlink

PAIR PROGRAMMING, XP, AND DISTRIBUTED COLLABORATION

Increasingly, programmers are working in geographically distributed teams. Escalating trends in 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 a programmer more effective than if all the programmers were, indeed, colocated. 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. The

technique is based on an emerging software engineering methodology known as *pair-programming* combined with nearly 20 years of widespread and active research in collaborative software systems. We aim to show that geographically distributed programmers benefit from using technology to collaborate synchronously with other programmers. Our objective is to demonstrate that the geographically distributed programmers who collaborate synchronously with other programmers will outperform geographically distributed programmers who work independently.

Professional interest in pair programming has risen dramatically in recent years with the success of an agile software development process called *Extreme Programming*, or *XP* [10,11] developed by Kent Beck. XP is distinguished from more traditional development processes by emphasizing (even requiring)

- pair programming
- test-first code development at the unit level
- full regression test support (with JUnit [12,13])
- lack of up-front detailed design
- frequent code refactoring [14]
- on-site client
- expectation of requirements changes.

XP practitioners develop requirements conversationally with the client, and deliver frequent working prototypes for feedback and changes. No code is written unless it is needed (no programming for the “perhaps” future), and when a design grows to the point that it feels unwieldy it is refactored and re-architected before further extension. The process is best known, though, for pair programming, which is at the heart of the productivity increases many XP teams are seeing.

Anecdotal and statistical evidence indicate that pair-programming—*two* programmers working side-by-side at *one* computer, collaborating on the same design, algorithm, code or test—is highly productive. Cockburn and Williams have produced statistical results that show pair-programmers produce higher-quality products in essentially half the elapsed time as individual programmers [8]. We believe the pair-programming model can be modified for distributed

collaborative development, and that we will see similar benefits.

To evaluate the effectiveness of distributed pair development, we are running controlled empirical studies involving students at North Carolina State University and the University of North Carolina at Chapel Hill. Students use interactive information technology over the Internet, such as *PCAnywhere* and *NetMeeting*, to jointly and simultaneously control a programming session and to speak with each other synchronously. Our early experiments have led us to develop the more video-enhanced environment described in the second half of this paper to support the remote synchronous collaboration required in DXP software development.

PREVIOUS WORK

Pair Programming

Pair programming is a style of programming in which *two* programmers work side by side at *one* computer, continuously collaborating on the same design, algorithm, code or test. One of the pair, called the *driver*, is 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 is, the greater the need for two brains [1, 6].

Controlled studies have shown that pairs finish in about half the time of individuals and produce higher quality code. The technique has also been shown to assist programmers in enhancing their technical skills, to improve team communication, and to be more enjoyable [1, 6, 7, 8].

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 [9]. 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 [3].

A primary consideration in virtual teaming is that of communication [4]. Poor communication can cause problems like inadequate project visibility, wherein everyone does his/her individual work, but no one knows if the pieces can be integrated into a complete solution. Coordination among the team members could also be a problem. Finally, the technology used must be robust enough to *support distributed development of software*, not just to provide communications.

The experiments we ran used virtual teams of two kinds. One kind was developing software in traditional, individual programmer style, and thus needed primarily communications support for integration of work; the other was developing software using the technical approach of *pair-programming*, where it is essential for proper collaboration that each team member have the illusion of sharing a single PC in real-time, synchronously.

Collaborative systems and distributed workgroups

The earliest example of a collaborative computer system was NLS-Augment by Engelbart [15], an initial version of which was demonstrated in the early 1960's. Engelbart's system used shared CRTs, audio connections, mouse, and keyboard to allow crude teleconferencing and shared examination of text files by users who were not co-located. From these early beginnings, collaborative software systems became the subject of widespread research more than 15 years ago, with the creation of the PC. Ongoing research tends to focus in three main areas: hardware to provide effective communications; software concepts that allow sharing of artifacts; and conceptual models of how people want to — or are able to — interact effectively.

Many collaborative system ideas developed in research labs have now found their way into practice, and we are seeing commercially viable products and services for supporting collaboration. These products are used to create virtual workplaces and allow people around the world to work on coordinated group efforts. The most commonly used collaborative tools are "chat rooms". Though simple in technical terms, chat rooms' wide usage demonstrates that *simple technology can be used very effectively*. Many commercial vendors support chat servers for exchange of text, audio, and video streams (Yahoo, Excite, PalTalk, and Microsoft to name a few) and hundreds of thousands of people use these services. Most usage of chat servers is for social interactions like game playing, but they are also used to support businesses. *We repeat that simple technology often gets you large gains*.

Shared software artifacts

Numerous researchers have developed software for supporting human interactions within shared artifacts. These

include general systems, like shared whiteboards for drawing, and shared editors for documents and multi-media streams. They also include special-purpose systems like ICICLE [16] for code inspection and review, and gIBIS for design review and logical argument construction [17]. Trellis [18] and MMM [19] are two systems for collaborative hypermedia and Web browsing. Trellis was built on the idea of structuring a hyperdocument as an abstract parallel process specification, allowing group interaction on the document to be defined in the document link structure. MMM extended this idea by extracting the process definition out of the document and allowed the group interaction rules (collaboration protocol) to be dynamically defined (as opposed to being hard-wired in the source code). In this way, group interactions can be developed and changed without changing the MMM software. Further research has led to methods for verifying the correctness of collaboration protocols [20,21].

In the experiments we have done, the shared artifact is, at the lowest level, the entire PC screen. We wanted to explore the technological feasibility of distributed pair programming with the simplest of technical structure. Thus we chose not to experiment with shared artifacts at the code or document level, allowing the programmers to work in pairs with exactly the same tools they use when programming alone.

Hardware and graphics for collaborations

Many of the most visible developments in collaborative systems have come from computer graphics. While their display technologies have remarkable sophistication in their visual imagery, most of them do not support *distributed* collaborations. The CAVE [22] is a multi-person, room-sized, high-resolution, 3D video and audio environment developed at the University of Illinois at Chicago. It functions as virtual reality theater, made up of three rear-projection screens for the front, right and left walls and a down-projection screen for the floor. Multiple users may sit in the space, wearing special glasses to decode the stereo projections. The CAVE is state-of-the-art in terms of visual impact and virtual reality presentation; it is limited in distributed collaboration support as it requires all participants to be present in the same space in order to work together. The DataWall [23] at MIT and the PowerWall [24] at the University of Minnesota are similar large display projects. The primary purpose of these large displays is to visualize and display very high-resolution data, often from large scientific simulations performed on supercomputers or from high resolution imaging applications. The large display areas allow small groups of collaborators to see the display clearly and without obstruction. It is possible to walk up to the display and point to features of interest, just as one would do while discussing work at a blackboard. However the emphasis is on graphics clarity, and not remote collaboration.

Unlike the previously mentioned display projects, Virtual Workspace [25] was intended as an environment to enable distributed collaboration over a network. It depends heavily on computer-generated graphics and virtual reality devices as well. ClearBoard [26] was similarly a non-co-located

collaboration support system that allowed two users to appear to sit face to face, and see the shared work between them; emphasis was placed on drawing applications. The system required special hardware (the clear screen), and was not built with COTS technology (as is our environment). Experiments using ClearBoard showed that *increased eye contact and the sense of presence of the remote collaborator was important in providing effective work performance.*

The *Office of the Future* project [27] at UNC, under the direction of Henry Fuchs, seeks to combine network-based collaboration with the superior graphics and image manipulation capabilities of virtual reality systems. It is a long-term project that will not be generally usable for years and it will require expensive special-purpose support hardware and high-performance graphics engines.

INITIAL PLATFORM EXPERIMENT

An initial experiment was done in early fall 2001 between NCSU and UNC to determine an effective technical platform to allow distributed pair programming. We wanted to use simple COTS technology – something that would be readily available to anyone – rather than research projects or platforms. Two pairs of programmers worked over the Internet to develop as a 4-person team 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 trailed 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 trailed 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 Webcams 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 Webcams.

Our goal was not to test if a remote pair could be as efficient as a co-located one, but to simply 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.

COMPREHENSIVE PAIRING EXPERIMENT

Hypothesis

After the platform experiment, we ran an experiment to assess whether geographically distributed programmers benefit from using technology to collaborate synchronously with each other. Specifically, we examined the following hypotheses:

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

The Experimental Procedure

The experiment was conducted in a graduate class, *Object-Oriented Languages and Systems*, 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. 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 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 co-located 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 (VBEE students had 37).

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

1. *Colocated team without pairs (9 groups)*

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

2. *Colocated team with pairs (16 groups)*

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

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

3. *Distributed team without pairs (8 groups)*

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

4. *Distributed team with pairs (5 groups)*

This fourth set of teams developed the project by working in pairs over the Internet. At the end, they integrated the various modules. They worked with the PC-sharing platform we detailed earlier.

In order to record their progress, the students used Bryce [5], a Web-based software-process analysis system used to record metrics for software development. Using this 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.

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.

EXPERIMENTAL 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 co-located 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.

Productivity

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

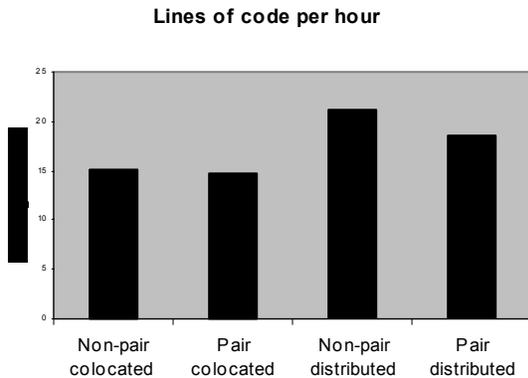


Figure 1

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

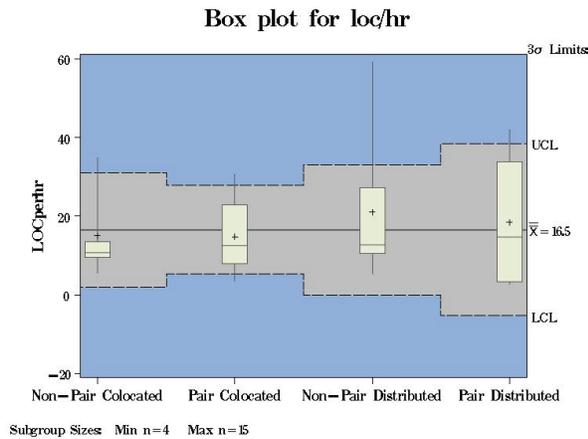


Figure 2

A box plot shows the distribution of data around the median. The vertical rectangle for each category shows the distribution of the middle 50% of the readings. The horizontal line inside each rectangle shows the median value for that particular category. The line segment from the top of the rectangle shows the range in which the top 25% of the values lie. Similarly, the line segment below the rectangle shows the range in which the bottom 25% of the values lie. Thus, the ends 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.

Quality

The quality of the software developed by the groups was measured in terms of the average grade obtained by the group out of a maximum of 110. The graph below indicates that the performance of students did not vary much from one category to another.

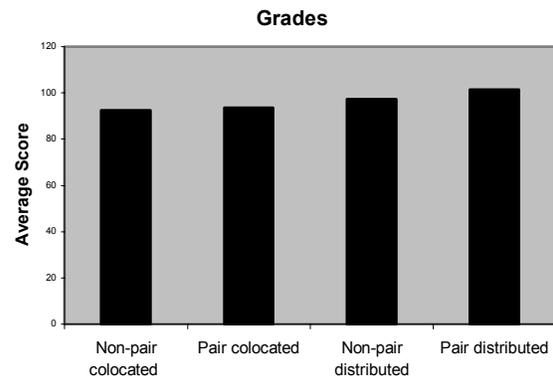


Figure 3

A box plot for the grades only corroborates the claim made above. Although nothing statistically significant can be said about the grades for the four categories, it is interesting to see that those teams performing distributed pair programming were very successful in comparison to other groups. The results of the statistical tests indicate that teams involved in virtual teaming were not significantly better than the distributed teams using pair programming, in terms of grade.

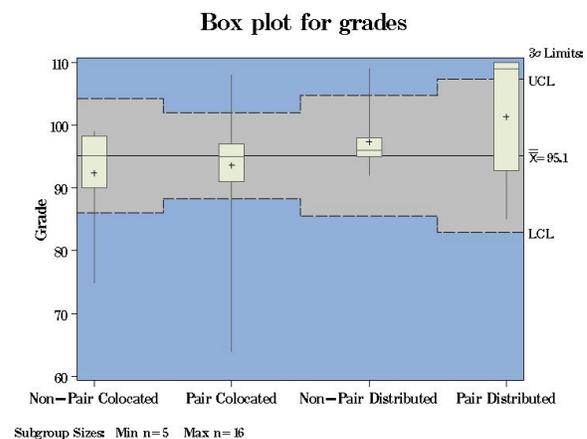


Figure 4

Student Feedback

Productivity and product quality is important. However, we also ran a survey to assess students' satisfaction with their working arrangement. Five out of the six students involved in distributed pair programming thought that technology was not much of a hindrance in collaborative programming. Also, 23 out of 28 students involved in virtual teaming with or without pair programming felt that there was proper cooperation in the team, meaning that the technical platform provided the facilities needed.

A COMPREHENSIVE ENVIRONMENT FOR DISTRIBUTED PAIR PROGRAMMING

We now shift focus from the experiment we have completed with a simple technology base to a description of the more comprehensive collaboration support environment we are constructing for DXP. Based on the results of earlier investigations into remote work systems and tele-presence, we are constructing a pair-programming station that will use hypermedia-augmented video projection to give the collaborators a better sense of "being there" while developing software jointly. We first give some background on previous research in this area, and then describe the hardware and software components of our DXP environment.

"Office of Real Soon Now" and VideoWindow

Even with the remarkable graphics technology available at the high end of virtual reality systems, much remains unchanged from Englebart's early prototype. The dominant paradigm of human-computer interaction in the workplace — a single user sitting in front of a single display with limited resolution and a WIMP interface, attached to a single computer — remains in force and has become a barrier to effective communication and collaboration for group interactions. The research we are doing is trying to break that barrier with relatively simple technology, compared to the systems just described.

We are building a more effective collaborative environment for pair programming using the results of some simple wall-size display experiments at UNC [28,29]. Whimsically termed the "*Office of Real Soon Now*" (a play on the name of the "*Office of the Future*"), it aims to get some of the benefits of large screens without waiting years and spending large amounts money. In this project, Bishop and Welch have produced double-width wall-sized displays for their offices using COTS projectors, video cards, and PCs. For their experiments they completely abandoned CRT displays and used only projected wall displays; after 3 years neither has any intention to return to CRTs. Benefits of the large wall displays include greatly reduced eye-strain; better interaction capabilities with students when discussing joint work; and expanded screen real-estate. Their experiments have concentrated on individual and co-located group use of the wall display technology, and have not involved networked collaborations.

Just as the "Office of Real Soon Now" seeks to have large-display benefits well in advance of the Office of the Future, we seek the "real soon now" benefits for distributed pair programming by using inexpensive COTS projection equipment, and ubiquitous PCs. The design of our DXP environment employs the principles uncovered at BellCore in the VideoWindow project [30]. In this experiment, two rooms in different buildings at BellCore (coffee lounges) were outfitted with video cameras and wall-sized projections. In essence, an image of one lounge was sent to the other and projected on the back wall, giving the illusion in each room of a double-size coffee lounge. The researchers discovered that many users found the setup to be very natural for human communication, due to its size. Two people, one in each room, would approach the wall to converse, standing a distance from the wall that approximated the distance they would stand from each other in face-to-face conversations.



Figure 5: DXP Collaborative pair programming setup

The DXP Environment

Our current experiments in distributed pair programming between UNC and NCSU have shown that programmer communication via a 19" to 21" display, while effective enough to allow development of good software, result in interactions that are somewhat stiff and limited in scope. The pairs so far have been given tools that support video interactions via webcam and postage-stamp-sized video windows. After initially turning the cameras on along with the shared PC and the audio, all pairs soon turned the cameras off to enhance performance. They reported that the video was too small to provide them with any communications enhancements.

We believe a far more effective collaborative environment can be created with a wall-sized display produced by projectors, and that a corresponding improvement in distributed pair programming will result from this enhanced video support for collaboration.

The equipment package needed for one office is:

- High-resolution video projector2 (2)

- Firewire camera + PCI video capture card (1)
- PC video card to handle two screens seamlessly (1)
- wiring, cable, microphones, screen boards, etc.

The cost for a single office is about \$8,000. We are working with four packages, outfitting two offices at each of the two research sites (UNC-CH and NCSU). This arrangement allows “local” distributed pairing at each site over the LAN, as well as pairing across sites with a wider-area network. Each office has two projectors. One is primarily used for video display of the remote collaborator. The other is for display of the shared computer “screen.” We are starting with an L-shaped screen setup, with the collaborator video image to the side of the programmer and the computer display to the front. We have placed the camera next to the projection wall rather than on the workstation in order to present each user with a view of the other’s office, with a side-view of the collaborator in the foreground. We will experiment later with different user placements and screen arrangements.

Figure 5 shows half the setup at UNC. Visible are the two projectors pointed at right angles to each other, and one of the screens. As the developer sits, he sees the shared PC desktop projected ahead, and the collaborator projected to the left. To communicate with the collaborator the developer turns to the left and speaks to the screen. The camera location with the screen gives a nice impression of the pair being face-to-face. This mimics the head movement needed to look at one’s pair programmer when working co-located.

We are working to eventually mix the video imagery (allowing the collaborators to see each other) with digital display information (the source code being developed), but for the first realization we use one projector for PC display and one for camera/video display. Communication is via directional microphones placed in the vicinity of the workstations, so the participants will not be encumbered with headsets. Two distributed collaborators interact much like they do with local pairing; to talk, one will turn to the other (face the projection wall) and speak openly in the room. Since the camera is on the projection wall, the remote collaborator will have the impression that the communicator is looking at him or her. Each will see surrounding context and an image of significant size. The illusion created is a “joint office” with the video walls, much like the virtual coffee lounge of BellCore’s project *VideoWindow* mentioned previously.

Software Platform: Video with hyperlinking

In addition to this hardware environment, we are developing for DXP software tools to more effectively support interaction between distributed pair programmers while developing programming project artifacts

As experiments progress, we will seek to identify areas in which collaboration among the programming pairs would benefit from software support and to build any shared artifact tools we may need (editors, inspection tools, etc.). Our first experiments, though, are to determine the effectiveness of the simplest approaches, using the observation made earlier that *simple technology often reaps large benefits*. Thus our first

experiments have been with a single virtual PC obtained via NetMeeting, on traditional PCs. NetMeeting provides not only a shared desktop, but audio communications as well.

One novel aspect of the DXP software environment is integration of the video stream from the camera with *OvalTine*, a hypermedia tool we developed at UNC to allow embedding of hyperlinks in video streams. In the section following we give an overview of the structure and image analysis techniques used in *OvalTine* to do video hyperlinking.

Having hyperlinking capabilities in the DXP video widow gives collaborators unique tools for organizing software development. Potential uses include creating hyperlinks off words on the collaborator’s whiteboard, effectively making the video image a virtual page. A user can also attach a notepad to the collaborator herself (the face), so that ideas needing discussion can be noted as they are thought of; when pairs switch, face recognition software will allow the previously annotated information to be retrieved via a mouse click on the collaborator’s face. Another possible use is linking programmers’ images to the code they have most recently worked on, or are responsible for. Such a use would exploit the reason *OvalTine* was created -- to allow video streams to be properly integrated with textual and image-based hypermedia documents (i.e., web pages and databases).

OvalTine allows hyperlink annotations in both real-time streams and in stored video. The later capability means that for study of DXP itself, the video window can be captured, archived, and then marked up with hyperlinks via *OvalTine*. Researchers studying the collaborators will be able to form video webs from the various DXP sessions. We are sure there are other uses for hyperlinks that the programming pairs will discover during experimentation when the *OvalTine*-enhanced DXP environment is fully online.

OVALTINE: HYPERLINKED REAL-TIME VIDEO

One of the problems keeping video from being a fully first-class data component of hypermedia documents is the difficulty of treating the objects depicted in video as identifiable, linkable content. Rather, video tends to be manipulated as frames of pixels with no further subdivisions. When link markup is done on video streams, it is done manually frame-by-frame. We have been working with *OvalTine*, a system for tracking objects in video streams so that hypermedia link anchors can be associated with the objects in the video frames. The *OvalTine* tracking system can be used to do automated link markup of video streams. While our previous work with *OvalTine* presented object tracking in real-time streams [31], we have recently developed extended techniques for markup of stored (archived) data. Our results allow hypermedia structure to be generated and added to large digital libraries of video data.

Every current popular method for adding active regions to video requires manual selection of video objects, on a frame-by-frame basis. No research efforts in automation have yet made it into common practice in a widely used system such as those from Apple (QuickTime, [32]). By contrast, an automatable object tracking system is much more desirable,

both for real-time applications, and for the automated addition of hyperlinks to the vast amount of archived video currently in existence.

A good overview of the issues and technologies in current hypervideo systems can be found in class notes at Texas A&M [33]. In the terms defined in this taxonomy, we are working on a system for automatically specifying mostly spatio-temporal links in hypervideo. Hypercafe [34] is often cited in hypervideo discussions; however, it is a presentation system mostly and does not support the dynamic and automatic link anchor creation we are exploring in OvalTine. The Multimedia Systems Lab at IISc India is doing work on object tracking in MPEG streams [35]. This project seems similar in scope and goals to OvalTine. They are tracking object in an MPEG stream, where as OvalTine is architected to be modular and extensible to define tracking and linking concepts at an abstract level, and to be applicable to different image and video formats with minimal extensions. Most other systems, however, that apply to hypervideo involve manual anchor creation when authoring hypervideos.

Two tracking modes

We have previously reported on the OvalTine system [31] and discussed the various distributed system architecture issues involved in storing and serving video hyperlinks in a client/server implementation. This section presents our continuing work with the system, demonstrating the use of the basic real-time image tracking algorithms for use in automated markup of stored video data. In the terms of the taxonomy we outlined in [31] this is a *Server/Archived* scenario, and we have chosen to implement a *Manual* object selection scheme.

In our initial OvalTine implementation, we demonstrated real-time tracking of faces in live video streams, such as the one from the collaborator camera in the DXP environment (an example of real-time videoconferencing). This tracking allows hypermedia link anchors to be associated with objects in the video window, creating a first-class hypermedia capability for video data. The face in the video frame becomes a live link, a selectable target for the user to click on to trigger some action. As the face moves around the video screen, the live target area moves with it, providing the illusion that the face itself is the hyperlink anchor. The target is optionally made visible by means of a simple highlight oval that moves with the face. Clicks within the oval count as a click on the face, and the enveloping hypermedia layer follows the link associated with the face. The links targets are URLs entered when the object is first selected as a link anchor (at the initiation of tracking); selecting one triggers the display of a web browser window with the proper URL loaded.

We have developed a second, equally important use for the OvalTine technology – automating the addition of link markup to stored (non-real-time) video data. There is increasing interest in video data being incorporated in hypermedia structures (which we will hereafter refer to as *hypervideo data*). Digital libraries are growing in popularity and scope, and video is an important component of such

archives. All major news services have vast video archives, valuable “footage” that would be of use in education, historical research, even entertainment. As noted earlier, the current best practices for link markup in video require completely, or considerably, manual markup of the video frames with the active, or hot, areas that serve as link anchors in hypervideo data. Broad access to vast caches of stored video “footage” will only be possible with automated link markup methods.

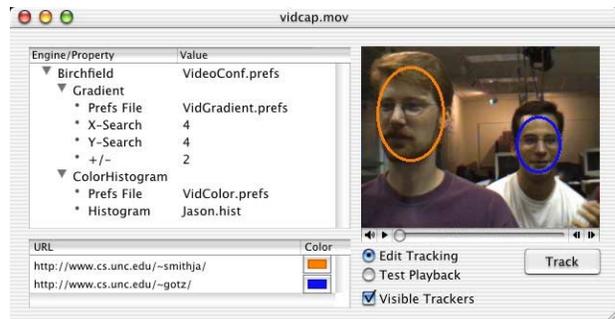


Figure 6. OvalTine tracker interface

We do not specify to what a link anchor may refer, nor do we limit the type of objects that may be designated as a link anchor. The goal of our work is to explore the creation and maintenance of hyperlinks in video streams, and to automate these procedures as much as possible.

User Interface Examples: Ovals, trackers, sprites, links, and multi-links

OvalTine is set up to allow link markup of stored video as an editing task. Figure 6 shows the video display window, the properties window, and the URLs window. The video display window presents the video data and shows the tracked objects as (optionally) outlined ovals. For real-time applications, such as video conferencing, the playback controls are inoperative. For markup of stored video, the user can do the standard *start*, *stop*, *pause*, and *slider* frame selection operations on the video stream.

The tracking properties window allows the user to select the type(s) of tracking algorithms to apply to the video frames. OvalTine’s architecture has been structured to allow multiple tracking algorithms to be chained together and applied in sequence. A user can even apply a different chain of trackers to each different oval if desired; OvalTine spawns a separate tracking thread for each oval. Some trackers work better than others in varying images; the selection of specific trackers to use depends on image properties such as color variability, background complexity, object motion, texture, etc.

Once a link anchor (oval) has been established in the video window, the user can associate one or more URLs with that anchor to be targets of the link(s). These URLs show up in

the URL window in the lower left. Figure 6 shows one URL for each oval, and the association is made by the color of the tag. For real-time links, mouse events are trapped in the video window, and the tracker information is used directly for the current frame to determine which stored URL is to be activated.

Stored video has an extra layer atop the base video data. During markup, the video stream is played and tracked as if in real-time. The areas that the trackers identify as active link anchors in each frame are captured and stored in a Sprite layer for Apple Quicktime [32]. OvalTine then uses Quicktime to overlay the hotspot layer onto the video image during playback, and traps mouse clicks in these areas for processing through the associated URL information in OvalTine's data store.

To edit, the user selects a starting frame with the slider controls, and then designates one or more objects to be tracked. The video is started with the "track" button, and the tracker chain for each oval causes the link anchors to follow the objects as they move in the video frame. The sprite infrastructure captures the layout information needed to maintain the link anchors in association with the video data. At any point the user may pause the video, add or delete ovals, and continue with tracking.

URLs can be added to the tracked objects at any point, either during tracking, or during playback editing of the marked-up video data. Though not shown, ovals can be linked to any first-class Web data, including another OvalTine video clip. There is also a "lost" link palette (not shown) that collects the URLs associated with objects that are being tracked, but move out of the video window. Any URLs associated with such an object are taken out of the URL window and saved in the "lost" list. This is a convenience that makes it easier for the user to re-associate these URLs if the tracked object should reappear in the video window and need to be tracked again.

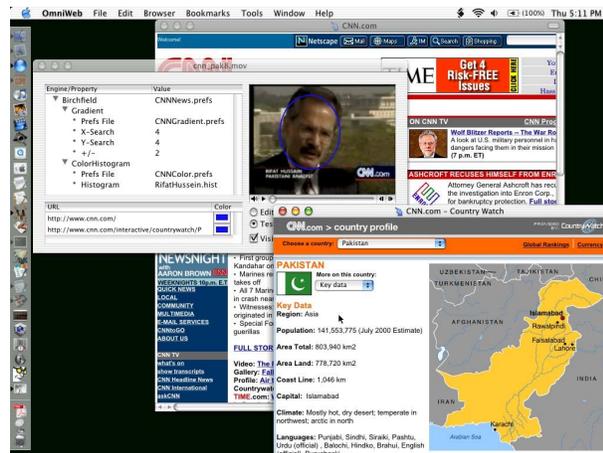


Figure 7. OvalTine application showing marked up video stream

Figure 7 shows OvalTine in use while viewing a marked-up video stream. Here we see a CNN clip where the face of the reported has been annotated with 2 different links. One link is to the CNN home page (seen displayed in the background). The other link is to an article and map on Pakistan, which is the topic of the video; this page is shown in the foreground.

CONCLUSIONS

The results of our experiment indicate the following:

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

Thus, the experiment is a first indication that distributed pair programming is a feasible and efficient method for dealing with team projects.

The successes of our simple DXP platform has led us to construct one that presents collaborators with a more significant video image, including the ability to create hyperlinks in a real-time video stream. Follow-on experiments in distributed pair-programming will be conducted using this video-enhanced DXP environment.

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.

This work was also supported with funds from NSF grant 9732577 and EPA grant R82-795901-3 to the Univ. of North Carolina, as well as equipment from IBM.

REFERENCES

- [1] L. A. Williams, "The Collaborative Software Process PhD Dissertation", Department of Computer Science, University of Utah, Salt Lake City, 2000.
- [2] J. T. Nosek, "The case for collaborative programming", *Communications of the ACM* 41:3, March 1998, p. 105-108.
- [3] S. P. Foley, "The Boundless Team: Virtual Teaming", <http://esecuritylib.virtualave.net/virtualteams.pdf>, 2000 Master of Science in Technology (MST) Graduate Symposium, Northern Kentucky University.

- [4] D. Gould, "Leading Virtual Teams", Leader Values, <http://www.leader-values.com/Guests/Gould.htm>. July 9, 2000.
- [5] <http://bryce.csc.ncsu.edu/tool/default.jsp>
- [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", Extreme Programming Examined, Succi, G., Marchesi, M. eds., pp. 223-248, Boston, MA: Addison Wesley, 2001
- [9] B. George., Y. M. Mansour, "A Multidisciplinary Virtual Team", Accepted at Systemics, Cybernetics and Informatics (SCI), 2002.
- [10] Beck, K., *Extreme Programming Explained*, Addison-Wesley, 2000.
- [11] Wells, J. D., "Extreme Programming: A Gentle Introduction," 2001, available on-line at <http://www.extremeprogramming.org/>
- [12] Beck, K., and Gamma, E., "JUnit Test Infected: Programmers Love Writing Tests," *Java Report*, July 1998, Volume 3, Number 7. Available on-line at: <http://JUnit.sourceforge.net/doc/testinfected/testing.htm>
- [13] Beck, K., and Gamma, E., "JUnit A Cook's Tour," *Java Report*, 4(5), May 1999. Available on-line at: <http://JUnit.sourceforge.net/doc/cookstour/cookstour.htm>
- [14] Fowler, M., *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 1999.
- [15] D. C. Engelbart and W. K. English, "A Research Center for Augmenting Human Intellect," presented at AFIPS Conference Proceedings of the 1968 Fall Joint Computer Conference, San Francisco, CA., 1968.
- [16] L. Brothers, V. Sembugamoorthy, and M. Muller, "ICICLE: Groupware for code inspection," presented at Proc. of Computer Supported Collaborative Work, Los Angeles, 1990.
- [17] S. Conklin and M. Begeman, "gIBIS: A hypertext tool for team design deliberation," presented at Proc. of ACM Hypertext '87, Chapel Hill, NC, 1987.
- [18] R. Furuta, P. D. Stotts, "Petri Net Based Hypertext: Document Structure with Browsing Semantics," *ACM Trans. on Information Systems (ACM)*, vol. 7, pp. 3-29, January 1989.
- [19] M. Capps, B. Ladd, and D. Stotts, "Enhanced Graph Models in the Web: Multi-client, Multi-head, Multi-tail Browsing," *Computer Networks and ISDN Systems*, vol. 28, pp. 1105-1112, 1996.
- [20] J. Navon, "Specification and Semi-Automated Verification of Coordination Protocols for Collaborative Software Systems Ph.D. Thesis," in *Department of Computer Science*. Chapel Hill, NC: University of North Carolina, 2001.
- [21] P. D. Stotts, R. Furuta, and C. R. Cabarrus, "Hyperdocuments as Automata: Verification of Trace-based Browsing Properties by Model Checking," *ACM Trans. on Information Systems*, vol. 16, pp. 1-30, January 1998.
- [22] D. Jones, "What is a CAVE TM?," pp. <http://www.sv.vt.edu/future/vt-cave/whatis/>, 1999.
- [23] MIT, "DataWall: Seamless, full motion ultrahigh resolution projection display," pp. <http://vlw.www.media.mit.edu/groups/vlw/DataWall-overview.htm>, 2000.
- [24] UMN, "Welcome to the PowerWall," pp. <http://www.lcse.umn.edu/research/powerwall/powerwall.html>, 1998.
- [25] H. Takemura and F. Kishino, "Cooperative Work Environment using Virtual Workspace," presented at Proc. of CSCW '92, Toronto, 1992.
- [26] H. Ishii, M. Kobayashi, and J. Grudin, "Integration of inter-personal space and shared workspace: ClearBoard design and experiments," presented at Proc. of CSCW '92, Toronto, 1992.
- [27] H. Fuchs, "The Office of the Future," pp. <http://www.cs.unc.edu/~raskar/Office/>.
- [28] G. Bishop, , pp. <http://www.cs.unc.edu/~gb/office.htm>, The Office of Real Soon Now.
- [29] G. Bishop and G. Welch, "Working in the Office of 'Real Soon Now'," *IEEE Computer Graphics and Applications*, pp. 76-78, July/August 2000.
- [30] R. S. Fish, R. E. Kraut, and B. L. Chalfonte, "The VideoWindow System in Informal Communications," presented at Proc. of CSCW '90, Los Angeles, 1990.
- [31] Smith, J., D. Stotts, and S.-U. Kum, "An Orthogonal Taxonomy for Hyperlink Anchor Generation in Video Streams using OvalTine," *Proc. of Hypertext 2000 (ACM)*, May, 2000, San Antonio, Texas, pp. 11-18.
- [32] Apple Computer, "Introduction to Wired Movies, Sprites, and the Sprite Toolbox", <http://developer.apple.com/techpubs/quicktime/qtdevdocs/REF/refWiredIntro.htm>
- [33] Francisco-Revilla, L., "A Picture of Hypervideo Today", <http://www.csdl.tamu.edu/~10f0954/academic/cpsc610/p-1.htm>, 1998.
- [34] Sawhney, N., D. Balcom, and I. Smith, "HyperCafe: Narrative and Aesthetic Properties of Hypervideo", Hypertext '96 Proceedings, ACM, Washington, D.C., 1996, pp. 1-10.
- [35] Multimedia Systems Lab at IISc, "Object tracking and hypervideo", <http://serc204a.serc.iisc.ernet.in/research/track.htm>