Potential and Actual Collaboration Support for Distributed Pair-Programming

Soporte a Colaboración Potencial y Real para Programación en Pares Distribuida

Alberto L. Morán[1], Jesús Favela[2], Raúl Romero[2], Hiroshi Natsu[3], Cynthia Pérez[2], Omar Robles[3] and Ana María Martínez Enríquez[4]

[1]Facultad de Ciencias, UABC, Ensenada, México
[2]Ciencias de la Computación, CICESE, Ensenada, México
[3]Code Services, Ensenada, B.C., México
[4]Departamento de Computación, CINVESTAV, México D.F., México
alberto_moran@uabc.mx, {favela, romero, cbperez}@cicese.mx
{hnatsu, orobles}@codeservices.com.mx, ammartin@ cinvestav.mx

Article received on July 31, 2007; accepted on October 30, 2007

Abstract
In Pair Programming – a software development technique that is part of Extreme Programming (XP) – two developers work side by side, on a single computer, to jointly produce an artifact. It has been reported that Pair Programming can be accounted for the development of higher quality software in half the time it required a single programmer. Pair programmers are meant to be collocated since they require continuous and fluid communication. However, the globalization of the software industry and the growth of open source software development are trends that motivate the need to support Distributed Pair Programming. For distributed pair programming to be effective, its support should facilitate not only actually collaborating in pairs, but also the pairing of distributed colleagues in an opportunistic and flexible manner. In this paper we present the COPPER collaborative editor, developed using the Collaborative Spaces Model, to support pair programming during actual collaboration, and for potential collaboration; awareness on the opportunities for collaboration when a distributed colleague is available or working on a file of common interest. We also present the results of an empirical evaluation of the actual collaboration features of the tool. The evaluation considered three different working conditions: pairs collocated sharing a single computer; distributed pairs using application sharing mode; and distributed pairs using collaboration aware facilities. In all three cases the subjects used the COPPER collaborative editor. The results support our hypothesis that distributed pairs could find the same amount of errors as their collocated counterparts. However, no evidence was found that the pairs that used actual collaboration awareness services had better code comprehension, as we had also hypothesized. Overall, according to comments from evaluation participants', COPPER improves Distributed Pair Programming, in subtle but significant ways, by allowing concurrent work and better maintaining awareness on the concurrent actions of the pair, in contrast to collaboration-transparent applications which do not support these features.
Keywords: Pair Programming, Distributed Software Development, Collaboration Awareness, Actual and Potential Collaboration, Empirical Evaluation.

Resumen
En la programación en pares – una técnica de desarrollo de software que forma parte de la Programación Extrema (XP) – dos desarrolladores trabajan uno al lado del otro, en una sola computadora, para producir conjuntamente un artefacto. Se ha reportado que la Programación en Pares permite el desarrollo de software de mayor calidad en la mitad del tiempo que requiere a un sólo programador. Los programadores en pares deben estar co-localizados debido a que requieren una comunicación continua y fluida. Sin embargo, la globalización de la industria del software, y el crecimiento del desarrollo de software de código abierto son tendencias que motivan la necesidad de dar soporte a la Programación en Pares Distribuida. Para que la programación en pares distribuida sea efectiva, el soporte debe facilitar no sólo la colaboración real en pares, si no también el establecimiento de pares de colegas distribuidos, y de una manera oportunística y flexible. En este artículo presentamos el editor colaborativo COPPER, desarrollado usando el Modelo de Espacios Colaborativos, para dar soporte a la programación en pares, tanto para la colaboración real, como para la Colaboración Potencial; la conciencia de oportunidades para la colaboración cuando un colega distribuido está disponible o trabajando en un archivo de interés común. También presentamos los resultados de una evaluación empírica de las características de la herramienta que dan soporte a la

colaboración real. La evaluación considera tres condiciones de trabajo diferentes: pares co-localizados usando una sola computadora, pares distribuidos usando una aplicación en modo compartido, y pares distribuidos usando una herramienta conciente de la colaboración. En los tres casos los sujetos usaron el editor colaborativo COPPER. Los resultados apoyan nuestra hipótesis de que los pares distribuidos pueden encontrar la misma cantidad de errores que sus contrapartes co-localizados. Sin embargo, no se encontró evidencia de que los pares que usaron los servicios de colaboración real tuvieran una mejor comprensión del código, como se había establecido en otra hipótesis. De manera global, de acuerdo con los comentarios de los participantes en la evaluación, COPPER mejora la Programación en Pares Distribuida, en formas sutiles pero significativas, al permitir el trabajo concurrente y al mantener de mejor manera la conciencia de las acciones concurrentes realizadas por el par, en contraste con aplicaciones que no soportan estas características.
Palabras clave: Programación en Pares, Desarrollo Distribuido de Software, Conciencia de Colaboración, Colaboración Potencial y Real, Evaluación Empírica.

1 Introduction

Pair Programming is a software development technique that is part of Extreme Programming (XP). XP is a lightweight methodology for small-to-medium sized teams developing software with highly changing requirements (Beck, 2000). XP promotes a discipline of software development based on principles of simplicity, communication, feedback, and courage. It achieves this by taking twelve commonsense practices to the extreme, which are: Planning Game, Small Releases, Metaphor, Simple Design, Testing, Refactoring, Collective Ownership, Continuous Integration, 40-hour Week, On-Site Customer, and Coding Standard, and Pair Programming (Beck, 2000).

In pair programming, two developers work side by side, on a single computer, to jointly produce an artifact (design, algorithm, code, etc.). It has been reported that this technique can be accounted for the development of higher quality software in half the time it required a single programmer (Cockburn and Williams, 2001; Williams et al., 2000).

The two programmers work as a unit, as a single mind responsible for all aspects of the artifact. One programmer, the driver, controls the pen, mouse, or keyboard to write the code. His colleague actively observes the work produced by the driver, looking for defects, alternatives and considering the implications of the strategy being followed. The pair changes roles periodically. Both participants are active throughout the process and share the responsibility for the work being produced (Williams and Kessler, 2000).

The successful application of this technique requires the use of an appropriate workplace: “The programmers should be able to sit side by side and program, looking simultaneously at the computer screen, sharing the keyboard and the mouse” (Williams and Kessler, 2000). While they work together, the couple should be able to share the keyboard without changing seats. Extreme programmers need to be constantly in touch with their team and for this, their workspace has to be open and facilitate communication among peers as well as casual and informal encounters.

Although there had been some criticisms against pair programming, mainly due to factors such as social dynamics (e.g. not everybody likes to program in pairs), lack of privacy (e.g. most programmers are willing to share their code, but not until it is ready), lacking of “quiet thinking time” (some programmers prefer to concentrate on a problem to solve it, rather than discussing it with a pair), and ergonomic issues (e.g. programmers can’t just adjust their work chair, monitor and keyboard to their most suitable position, as they should take into account their pair), [Stephens and Rosenberg, 2003], pair-programming remains a well accepted practice among the XP community.

An important trend in software development has been the globalization of the software industry (Herbsleb and Moitra, 2001). With increased frequency, software developers are required to work in groups that are geographically distributed. There are several motivating factors behind this trend. Among the most important ones (Herbsleb and Moitra, 2001):

1. Software companies require highly qualified human resources and they look to fulfill this need by hiring programmers in different cities and countries.
2. To be closer to the market and have a shorter response time, many companies have placed development groups closer to their client’s location.
3. Virtual development groups might need to be created quickly to exploit new market opportunities.
4. By working on different times zones development groups can work continuously on critical projects.

Under these circumstances the requirement for pair programmers to be in the same location seems an important limitation of this approach.

For this reason we have developed the COPPER synchronous collaborative writing tool, designed specifically to support pair programming among distributed collaborators (Natsu et al., 2003). In this paper we describe the main conceptualizations underpinning its design and implementation, as well as report the results of an empirical evaluation performed to determine the feasibility of distributed pair programming and the services offered by COPPER in this regard. It has been argued, for instance, that Open Source Software (OSS) development differs from agile software development methods, such as XP, mostly in the fact that OSS developers are geographically distributed while agile developers work in close (Warsta and Abrahamsson, 2003). If XP techniques, such as pair programming, can be supported to be done at a distance, they could be adopted by the growing community of OSS developers and other distributed software development organizations.

The paper is organized as follows: Section 2 discusses the need for distributed developers to be able to team opportunistically. While previous work on distributed pair programming has focused on the support for actual collaboration (production of artifacts), we argue that identifying the opportunities for collaboration through awareness of presence of colleagues, their availability, and the task being undertaken is necessary to support the light and agile development processes associated with pair programming. In Section 3 we present the COPPER pair programming environment that incorporates design features meant to provide collaboration awareness and allow developers to identify opportunities for collaboration. Section 4 presents the design of an experiment conducted to evaluate some of the features of COPPER and the feasibility of distributed pair programming. The results of this study are presented and discussed in Section 5. Finally, Section 6 presents our conclusions and some directions of future work.

2 Distributed Pair Programming

The absence of physical proximity, and thus that of a suitable physical workplace, introduces significant challenges for distributed pair programming (Kircher et al., 2001). These challenges could be addressed from two different perspectives: i) during the establishment of a pair programming session and ii) during an actual distributed pair programming session. Little has been said in the literature about how pair programming sessions ought to be established. It is clear that some coordination and negotiation is required to program pairing sessions, which could disrupt the working rhythm of participating developers. In a distributed development environment participants would require additional coordination efforts to set up pair programming sessions, which could get in the way of the nimbleness required by agile development methods such as Extreme Programming. The effort required to negotiate and establish a distributed working session might be too costly for a development environment that requires close coordination and communication, rapid feedback, and collective code ownership (Stotts et al., 2003).

2.1 Issues from Pair Development and Pair Programming Session Establishment

To motivate our later introduction of the opportunistic establishment of pair programming sessions as a requirement for distributed pair programming, let us start discussing how pairs are established in the traditional collocated case.

Evidence in the literature suggests that pairs can be established a priory or they can be established opportunistically. In the first case, pairs can be assigned by an authority in the group (e.g. a professor or a manager), and ideally they should work together constantly (Williams et al., 2000). However, in reality, this is not necessarily the case, as there are several situations that make it difficult for colleagues to join in a pair programming session (e.g. illness, time conflict, efficiency, etc.). Furthermore, it is argued that when the whole development group adopts pair programming as the normal way of working, the long term continuity of any particular pair becomes less important. The ideal becomes then having a pair, not having the pair, for all development. Also, by pairing regularly with the other members of the group, individual programmers maintain sufficient general awareness to opportunistically become a required partner, or a substitute for a missing one. This leads us to the second case, where although usually you look for someone in particular when starting a task; more commonly, you just find and pair with someone who is available (Beck, 2000, p. 100). An example from an academic situation that privileges or even makes mandatory having a pair, instead of having the pair is provided in Cockburn and Williams (2001):

"Let's talk about pair programming […] Pair programming is mandatory. All production code must be written with a partner present. […]; *What if I need to write code and my partner isn't available?* […] Then you find someone else. One of the goals is to spread knowledge around […]; *What if there's no-one else around?* […] if there really is nobody around, push your keyboard away and wait".

From the previous discussion, we notice that pairs, either defined a priori or opportunistically selected, require being aware of their whereabouts and general activities in order to start and enter into a pair programming session effortlessly. In the collocated case, physical proximity allows them to look at each other and to mutually determine a right moment to negotiate or actually start a pair programming session. This is performed based on the information that is readily available in the physical shared environment. However, this is not necessarily the case for distributed pair programming. The main reason for this is that being distributed, pairs cannot interact in the ways they are used to, and that things that are taken for granted in traditional face-to-face (physical proximity) situations (e.g. talking to each other, making eye contact, gesturing and pointing at each other), are not possible, very difficult or not well provided through current supporting applications.

In this regard, Kircher et al. (2001) identified that Distributed eXtreme Programming (DXP – respective to Distributed Pair Programming) assumes the existence of certain conditions, tools and technologies (i.e. connectivity, e-mail, configuration management, application sharing, video conferencing and familiarity) which relaxes the requirements for physical proximity made by Pair Programming. However, this introduces challenges that DXP must face, concerning: Communication, Coordination, Infrastructure, Availability [negotiation], and Management.
Key to our discussion are the challenges related to communication, coordination and negotiation of availability.
Concerning communication, being distributed, it is difficult for the participants in the pair to obtain information on how each other react to what the other says or does. Thus, to judge a reaction, information on body gestures, the face, or even the tone of the voice of the other person is required.

Concerning coordination, being in two different locations, the challenge consists on how to synchronize their availability, adjust the time differences, and coordinating distribution as well as the integration of activities and results of the activities. In addition, sharing documents and applications can become a significant challenge.

Finally, concerning availability, when distributed, people may be available at different times. This can be the result of people working on multiple projects, due to personal limitations, or due to their being in different time zones, an inherent feature of global software development. Some solutions proposed to address these challenges (e.g. Kircher et al.; 2001; Herbsleb and Mockus, 2003) mainly consist in establishing pre-determined session times (planned interactions – e.g. "members in different locations could exchange daily e-mails containing their schedules for the day or use shared calendar applications, and by assigning certain slots within the day to work on the project"). Thus coordination and availability are negotiated using asynchronous tools, or by convening to establish contact by phone or video conferencing at pre-determined times (it should be noted that both the phone and video conferencing require the people involved to be available to establish communication (Tang et al., 1994)).

One of the main problems with these solutions is that communication, coordination, and availability negotiation are restricted to planned or scheduled communications (interactions), thus constraining the way in which people might get into collaboration or production (i.e. pair-programming), and therefore, disregarding the use of informal interaction to establish contact, and from which, eventually, get into collaboration.

An additional solution proposed by Herbsleb and Mockus (2003) is the "judicious" use of tools supporting informal communication and presence awareness (e.g. Instant Messaging). They argue that these tools have the potential to substantially lower the difficulty and frustration associated with contacting a remote pair. They also argue that this is possible by allowing a pair to time his/her communication attempt for a moment when the probability of the other being available to reply is much higher (e.g. someone "arriving" or getting online).

Taking into consideration the previous evidence, we argue for that need to go further, by not only providing support for informal interaction, but also, providing support for opportunistically starting or getting into collaboration in pair programming sessions to actually perform work.

For this reason, we propose to provide support not only i) to actually collaborate once a pair-programming session has been established, but also ii) to identify the possibility of collaboration with a pair, iii) to determine whether and when it is an adequate moment to collaborate for both pairs, and iv) to actually start a pair programming session. To achieve this, we introduce the concepts of Actual and Potential Collaboration, Potential Collaboration

Awareness and Actual and Potential Collaboration Spaces (Morán et al., 2004); and illustrate how they can be used to further facilitate the inclusion of support for starting or getting into a pair programming session in a distributed manner.

2.2 Actual and Potential Collaboration, Awareness and Spaces

Actual and Potential Collaboration represents complementary moments in the collaboration process. We consider Actual Collaboration as occurring while people work actively with others on a collaborative effort (e.g. in a pair programming session), mainly using a shared space representation (either physical or virtual) and mostly using synchronous and quasi-synchronous interactions, although asynchronous ones have been also considered. This kind of collaboration represents the major focus of the groupware development community, as most current applications, systems and models focus on providing support for it (Gaver et al., 1992; Bly et al., 1993; Morán et al., 2004). On the other hand, we consider that Potential Collaboration, in opposition to the former, refers to the possibility of collaboration, and as such occurs while people perform work individually, not necessarily in relation with a collaborative effort, mainly out of a shared space representation, and if in collaboration, mostly in (long-delayed) asynchronous mode. We believe that although an important preamble to actual collaboration, this kind of collaboration has been disregarded by most current models, systems, and tools that support collaborative work.

Potential Collaboration Awareness is a special kind of awareness specifically conceived to provide people with the required information to identify or create opportunities for collaboration with others, and to perform an informed decision on how and when to establish an interaction that may lead to Actual Collaboration. Thus, to better fulfill these needs, Potential Collaboration Awareness does not only include "general sense" information from the "informal" situation surrounding actual collaborators ("who is around and what they are up to"), but also more detailed information from the "formal" (individual or actual collaboration) situation in which potential collaborators are involved. This includes information on the relations among the people, artifacts, and activities involved, for instance:

- Who is present? Where?
- Which artifacts are involved? To whom are they assigned?
- Which activities are being performed? By whom? and
- Which is the degree of completion of these activities?, among others.

Actual and Potential Collaboration Spaces represent specialized conceptualizations and characterizations of space specifically conceived to provide support for Actual and Potential Collaboration, respectively. On one hand, Actual Collaboration Spaces are designed to efficiently support the three axes of ongoing collaboration proposed by Calvary et al. (1997): Communication, Coordination, and Production; and as mentioned earlier, represent the major focus of the groupware development community. On the other hand, Potential Collaboration Spaces are designed to efficiently take advantage of the potential for collaboration by providing users with the appropriate information for i) identifying opportunities for collaboration, and for identifying the appropriate moment to establish an interaction, as well as with the adequate mechanisms to ii) actually establishing the interaction that may result in actual collaborative work, and thus, getting into collaboration (Morán et al., 2004).

In the next section, we illustrate how these concepts were applied in the design and implementation of COPPER (Natsu et al., 2003), a tool to provide support for distributed pair programming, not only during actual collaboration (i.e. the actual pair programming tasks), but also for potential collaboration (i.e. to opportunistically identify and start pair programming sessions).

3 The COPPER Pair Programming Environment

COPPER is a synchronous collaborative environment designed to support pair programming (Natsu et al., 2003). It is based on a client-server architecture and composed of two main subsystems implementing potential and actual collaboration spaces, respectively. The former is implemented as a User and Document Presence (U&DP) service, and the latter as a Collaborative Editor, providing application support for potential collaboration (e.g. be aware of the

potential of collaboration, and take advantage of this information to determine whether and when to start actual collaboration) and for actual collaboration (e.g. write programs, access document services, and communicate with peers), respectively. The architecture of the system is presented in Figure 1.

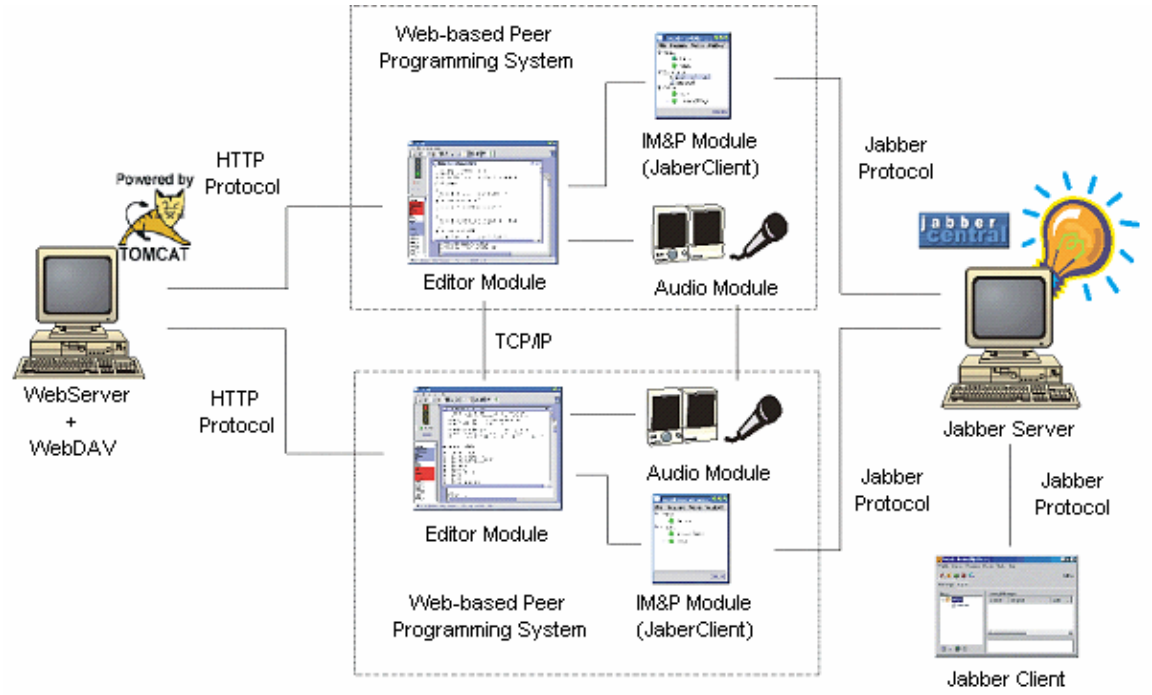

Fig. 1. Architecture of COPPER

3.1 Potential Collaboration Space: The User and Document Presence Service

The User and Document Presence (U&DP) subsystem is implemented as an extended Doc2U client (Morán et al, 2001). It extends the functionality of traditional Instant Messaging and Presence (IM&P) systems (send and receive messages to/from collaborators, manage the user's own presence and provide this presence information to other collaborators), to provide first class presence to documents stored in the Document server, and to allow interaction with documents in a similar way as interacting with users (Morán et al, 2001). This includes: adding or deleting documents from the document (presence) list, and sending "group" messages to subscribed users of a document. Furthermore, subscribed users receive messages from the U&DP service whenever the document's availability and status information changes. Potential Collaboration Awareness information elements, and additional functionality of the IM&P module, are accessible by means of several components, as shown in Figure 2.

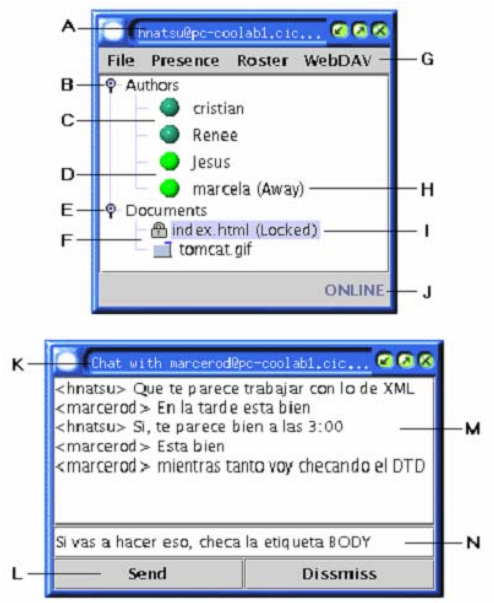

Fig. 2. Doc2U client of COPPER's User and Document Presence Module

The "traditional contact list" is decomposed into an author list (Figure 2B) and a document list (Figure 2E), which are managed by means of operations in the Roster menu (third option in 2G). These lists show availability and status information of subscribed users and documents. Availability options for users include "Online", and "Offline", while for documents they include "Available", "Locked" or "In-use", and "Not available". These options are shown by means of availability icons (Figures 2C, 2D, and 2F). Additional status information is shown by means of labels associated to the list elements (Figures 2H and 2I). Status options included are "Online", 'Away", "Extended away", "Free for chat" and "Do not disturb", which can be specified by means of the Presence menu (second option in 2G). Own identity and status information are shown by means of the application's title bar (Figure 2A) and status bar (Figure 2J), respectively. Document operations are available in the WebDAV menu (fourth option in 2G). Using these operations a user can put a lock on a document, get it to work on it locally (e.g. using the Editor module), save it back to the Document server and finally unlock it to allow other users to work on the new version of the document. Lastly, concerning communication, the module allows sending messages to users directly or through documents. Sending messages through a document allows a form of "group" message to all users subscribed to that document. Basically, two types of messages can be sent or received, chat and instant messages. Chat messages allow "concatenating" several messages on a single window to present a "conversation", while instant messages allow sending one message at a time. As users can establish several chat conversations or receive several instant messages from different users, the name of the user with whom the conversation is held or whom sent the message appears in the message window title bar (Figure 2K). An example of chat message window containing an ongoing conversation is shown in Figure 2M. The user types a new message to be added to the conversation (Figure 2N) and sends it by pushing the "Send" button (Figure 2L).

3.2 Actual Collaboration Space: The Collaborative Editor

The editor can be used disconnected from each other (in individual mode) or connected in pairs (in synchronous collaborative mode). Users can be either co-located or distributed on the Internet. The editor implements a turn-taking synchronous editor (see Figure 3). The user holding the floor (or editing right) uses the editing window (Figure 3F) to work on the currently loaded document.

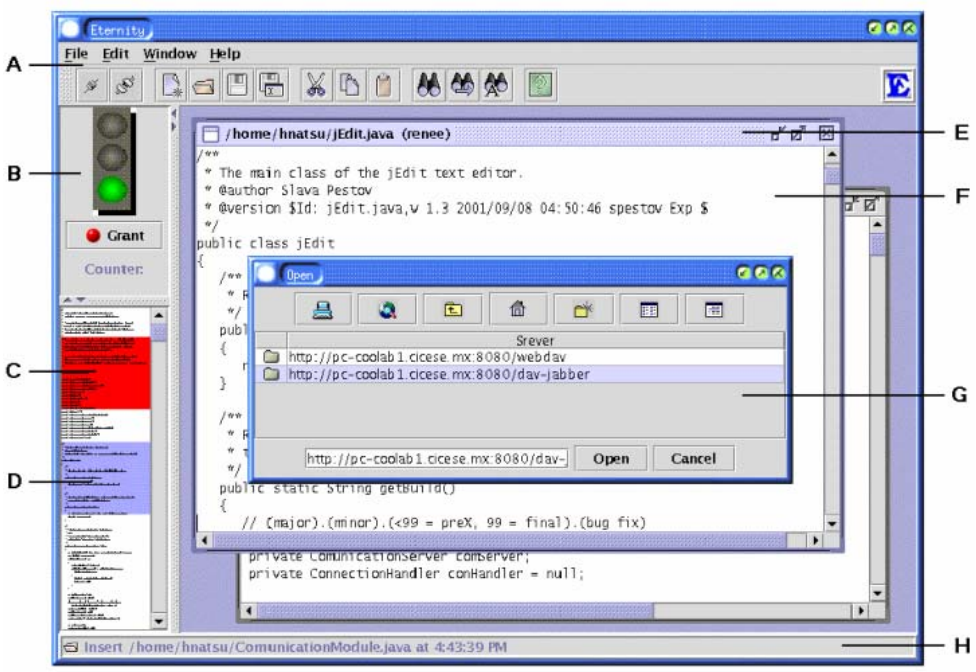

Fig. 3. The COPPER Pair Programming System

Common document management and editing functionality is provided by means of the application‘s Menu bar and the toolbar (Figure 3A). Actions performed in the editor are propagated to the collaborator’s client. As an example of their use, consider the Open button (third button left to right) of the toolbar. When pressed, an Open dialog (Figure 3G) appears, providing access to documents stored in the Document server. This dialog presents an integrated file hierarchy with documents from the local machine and from other distributed WebDAV servers (Whitehead, 1998). This allows for seamless navigation and document retrieval from the individual or collaborative work environment. Several documents can be edited at the same time, even if these documents come from different WebDAV servers.

The Document server offers a centralized information repository, which provides document storage, editing access control, user authentication and permissions to avoid unauthorized accesses, and document presence extensions through an instant messaging client, which “listens” and informs the U&DP system the results of operations performed on the documents (Morán et al., 2001).

Each editor client “owns” the documents opened by its user, and it is the only one allowed to perform operations, such as Save, Save As, and Close, on these documents. When the connection is broken, each client keeps their “own” documents, and the user can continue working on them in individual mode. While in this mode, clients are ready to send or receive invitations to begin collaborating. Floor management (or editing access control) is represented using a “traffic light” metaphor (Figure 3B), which is activated when working in collaborative mode. This component includes an action button to request and grant floor control.

Actual collaboration awareness is provided by means of a radar view (Figures 3C and 3D), editing window titles (Figure 3E), the status bar (Figure 3H), and the floor or editing control access component (described earlier). The radar view provides a general overview of the document being edited; it shows document changes in real time, and serves as a document navigation tool; selecting a particular line of the document in the radar view causes the current editing window to display the segment of the document where the selected line is present. Editing window titles (Figure 3E) display the name and location of the document, as well as the identities of the owner of the

document and of the collaborator (if present).

The status bar (Figure 3H) provides information on the last operation performed by any of the collaborators, as well as on the date and time it was performed. Finally, communication during actual collaboration is achieved through the same chat and instant messaging tools used to establish initial contact and communication.

3.3 Collaboration Awareness in COPPER and Pair Programming

As described above, COPPER's actual collaboration space (editor) is a collaborative aware application with a relaxed WYSIWIS interface (Ellis et al., 1991). Previous empirical work on distributed pair programming (e.g. Stotts et al., 2003; Baheti et al., 2002) has used application sharing tools, such as MS NetMeeting, to support the programming task. NetMeeting is a collaborative transparent application, that is, it provides limited awareness of the presence, actions and intensions of collaborators. In addition, NetMeeting supports a strict WYSIWIS (What You See Is What I See) interface, in which both participants have the same view of the shared application. COPPER's main differences with respect to the use of application sharing can be summarized as follows:

1. **Concurrent Work.** In COPPER collaborators can browse through the source code independently. This allows the observer for instance, to move to the top of the file and double check the name of a variable or method, while the driver continues writing the code. It also allows them to open another file or browse a manual to support the task at hand without having to leave the collaborative environment, or interrupting the colleague.
2. **Double Pointer.** With application sharing only one collaborator controls the whole application at a time, write code, scroll the document, or select part of the code while his colleague only observes the results; whenever the observer moves his pointing devices he gains control of the floor. In contrast, in COPPER the observer can at any time select part of the text to indicate an area of interest to his colleague, or scroll his window to look at a different part of the code, without affecting the view of the driver.
3. **Radar View.** Since the views of both collaborators are not necessarily synchronized in COPPER, a radar view is provided to give them awareness of the location of their pair and facilitating their movement throughout the file. Thus, if the observer calls the attention of the driver to a possible coding error, the former can quickly synchronize his screen with that of his colleague by using the radar view.
4. **Support for Potential Collaboration.** COPPER allows programmers to discover opportunities for collaboration and thus seamlessly move back and forth from pair programming and individual programming as needed. This is achieved through its user and presence awareness module.
5. **Awareness of Floor Control.** Using the traffic light component, COPPER provides information on the state of a request to gain the floor control. This way, information transmitted via audio or chat messages following a social protocol ("Could you give me the control?; Yes, just let me finish this" or "could you drive now?; Ok, I am taking the control"), is augmented or reinforced by "acting" the request or grant action through the traffic light component. Thus, request and grant actions are "spoken out loud" while they are performed, allowing for a smooth transition.

4 Evaluation of COPPER

In order to explore the potential and limitations of COPPER's actual collaboration support for distributed pair programming, we conducted an evaluation experiment, both in terms of the results achieved as reflected in code quality, and the actual collaboration awareness features that the tool provides.

4.1 Experimental Design and Procedure

To guide our research we established two working hypothesis:

H1: Distributed pair programmers working remotely will find the same number of defects as collocated programmers in the same amount of time.

H2: Distributed pair programmers using COPPER will have better code comprehension as distributed pair programmers using application sharing in NetMeeting.

The first hypothesis aims at establishing that pair programming can be as successful when the group is distributed as when they are collocated, if appropriate technical support is provided.

The second hypothesis is established to assess if the additional collaboration awareness services provided by COPPER better support the intense level of interaction required for synchronous tasks such as software programming or design, when compared with NetMeeting.

The subjects of the study were 12 graduate students in computer science and proficient in the Java programming language. The students were randomly grouped in 6 pairs. All subjects attended a one hour session that included an introductory lecture on pair programming, an explanation of the features of the COPPER editor and time for hands-on experience with the system. In addition, the subjects completed a questionnaire which focused on their previous programming experience.

We used a within-subjects design; with all six groups asked to perform three different programming tasks, each of them with a different setup.

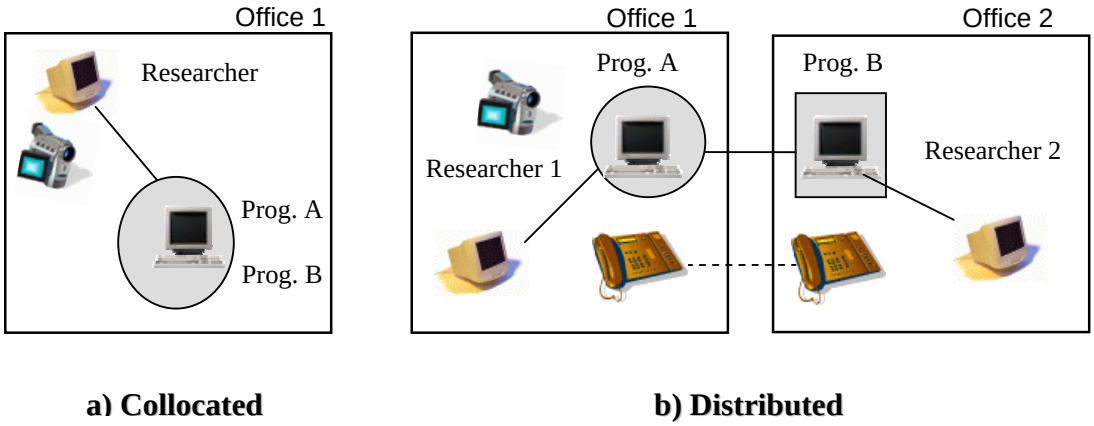

Fig. 4. Physical layout for the experiment. a) Collocated condition. b) Distributed conditions (NetMeeting and COPPER)

4.2 Experimental Setup

The modality in which the subjects performed the task was our independent variable. The experiment required both subjects to collaborate in performing three different programming tasks; each task was performed under a different condition.

Collocated Condition. In this condition, the two programmers are in the same office and share a single computer running the COPPER editor in single-user mode. The programmers share the display and keyboard as in traditional pair programming sessions (Figure 4a).

NetMeeting Condition (collaboration-transparent). In this condition, the programmers are in different offices, each of them with a workstation (Figure 4b). The subjects had a voice connection through a telephone for the duration of the task. The telephone is left in speakerphone mode to free them from having to carry the handset. The subjects use the COPPER editor in single-user mode and share the application through MS NetMeeting.

COPPER Condition (collaboration-aware). The physical setup of this condition is similar to the NetMeeting condition (Figure 4b). The only difference is that rather than using NetMeeting to share a single application, they used the COPPER editor in collaborative mode. In this way, they had access to the collaboration awareness features of COPPER, such as the radar view, and the use of the semaphore for floor control.

The programmers were videotaped when performing all three tasks. In addition, there was a researcher in each office who was present at the time of the experiment. This researcher kept control of the time and recorded the number of times the programmers exchanged control of the floor and kept track of the errors identified, corrected, and introduced in the code. To track the progress of the programming team without looking over their shoulders, the researchers had a monitor in a separate desk connected to the programmer's workstation (see Figure 4).

4.3 Procedure

The pairs had approximately 55min. to complete each task, which was divided in five phases. The tasks were performed one after the other with 5 to 10 minute rest periods between tasks, for an approximate total duration of the experiment of 3 hours per couple. The tasks were executed as follows:

Phase 0. During 10 minutes the programmers were introduced to the programming task they had to perform and the condition in which they had to work. Each person was given an initial version of the program assigned for that task. Each program was injected with 10 errors of different types (syntax, assignment, logic, and interface). The subjects were told that the code included errors but neither their type nor the number.

Phase 1. For the next 15 minutes the pair introduced the code into the editor. To keep this time constant for all programming tasks part of the code was typed before hand, they were asked to type the last 60 lines of code. Previous trails indicated that 15 minutes was sufficient to complete typing the code. The pair was able to choose who will be the driver and who the observer. They could also interchange roles during the process at will. Although the objective of this phase was to introduce the code into the editor, we expected them to detect and correct some of the errors and become familiar with the code.

Phase 2. Once the code was typed in, the subjects were given a maximum of 15 minutes to correct the program, that is, to find and correct errors. This included errors that were introduced by the researchers, and at times, errors introduced by the programmers themselves during Phase 1.

Phase 3. In this phase the programmers were given a text describing a simple modification to be made to the code. They were given a maximum of 15 minutes to discuss the solution and modify the code as required.

Phase 4. Finally, the pair was asked to complete a survey with 12 Likert-scale assertions, which included topics such as their satisfaction with the modality in which they worked, the perception of the participation of their colleague, and their understanding of the programming task.

4.4 Experimental Tasks

The programming tasks were designed to be able to be completed in the amount of time that was provided so that the students would concentrate on correcting errors and delivering a high quality code. We decided against the alternative of assigning more demanding tasks that required several days to be completed, since it was not possible to have the level of control over the experimental conditions that we required.

To avoid learning effects the pairs worked on the tasks in different order. Each group was asked to perform the following three programming tasks on a different condition:

LOC. The purpose of this program is to count the number of lines of source code in a file. The program takes a source file as input and returns the total number of lines contained in it. In the last phase the programmers were asked to modify the program to eliminate the number of lines with comments from the final count and also report the total number of methods in the source code.

SORT. This program reads a file containing a list of numbers in random order and sorts them in ascending order using the selection sort algorithm. The modification required the students to use insertion sort and present the results in descending order.

N Figures. This is an object-oriented program that reads a file with data of geometric figures of different types and calculates their area, using the method appropriate to each type of figure. The extension requested was to add a new class with a different type of figure.

4.5 Measures

Since we kept constant the amount of time dedicated to the task, we concentrated on measuring the quality of the code since this is an area in which one would expect pair programming to be particularly useful.

To estimate code quality we measured the number of errors that were detected and corrected during the task. A researcher observed the programmers as they performed the task and recorded when and by whom errors were detected and corrected. She also indicated as well when new errors were introduced.

Figure 5 shows the form used by the researchers to track the debugging of the program. One form was filled for each programming task indicating the group, the time spent on each phase, the condition under which they worked, the turn changes, the time and phase when the errors were found and corrected, and observations from the researcher.

In addition to the number of errors detected, we measured code comprehension, since we hypothesize that the help of the colleague would help a programmer understand the code more easily and this might be negatively affected by distance and positively affected by being collocated or having the awareness tools provided by COPPER. It has been reported that pair programming leads to improved comprehension and learning of the project and unfamiliar topics (Sanders, 2002).

We measured code comprehension by whether or not the pair was able to successfully modify the code to incorporate the new functionality as required to complete the programming task, and through the questionnaire at the end of the task where they were asked if they understood the program and whether the help of his colleague helped in this regard.

5 Evaluation Results and Discussion

As stated earlier, we were interested in determining whether (1) Distributed pair programmers working remotely could find the same number of defects as collocated programmers in the same amount of time, and (2) Distributed pair programmers using COPPER could have better code comprehension than distributed pair programmers using application sharing in NetMeeting.

In order to answer (1) we looked at the quality of the program (code) produced by the programmers, while in order to answer (2) we looked at the code comprehension expressed by the programmers through a questionnaire at the end of the experiment, as well as on their ability to complete the tasks.
A brief presentation and discussion of the results concerning each point follows.

5.1 Program quality

The quality of the code produced by the programmers was determined by the number of errors detected and corrected in each modality.

Table 1 shows the number of errors detected by each team for each of the three programming tasks. The results are grouped by condition, with two teams per condition (A and B), per task.
The NetMeeting condition was the most productive in finding errors, with a total of 25, followed by the Collocated mode with 24 errors detected. In the COPPER mode 22 errors were identified.

An ANOVA performed on this data shows that there is no statistically significant difference in the number of errors detected in the three modalities ($F = 0.1211$, $df = 17$, $p < 0.05$).

These results provide evidence in support of our first hypothesis. That is, distributed pair programmers working remotely CAN find the same number of defects as collocated programmers in the same amount of time (H1 Accepted).

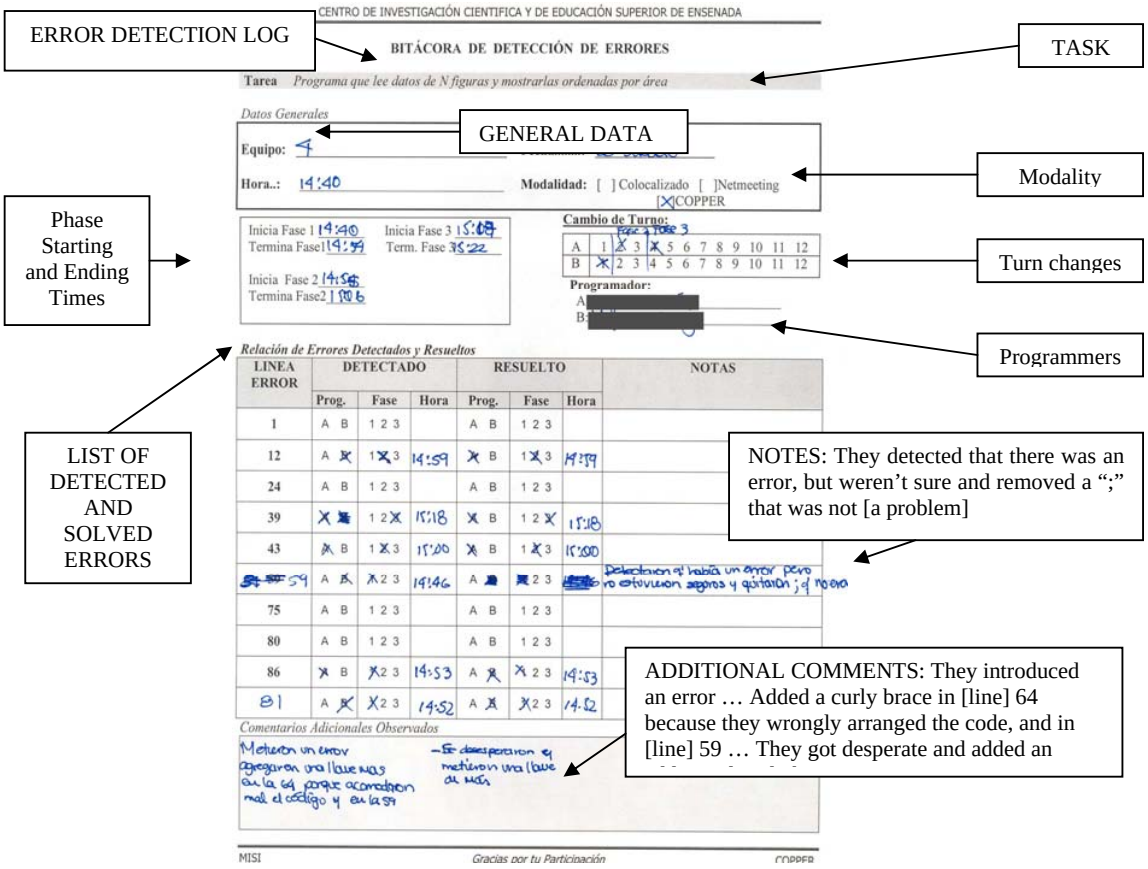

Fig. 5. Coding form used during the experiment

Table 1. Errors detected by each programming team

Condition		N Figures		LOC		SORT		Total
		Errors detected	Team	Errors detected	Team	Errors detected	Team	
Collocated	A	2	(5)	5	(1)	6	(3)	**24**
	B	5	(2)	4	(4)	2	(6)	
NetMeeting	A	4	(3)	5	(5)	7	(4)	**25**
	B	2	(6)	3	(2)	4	(1)	
COPPER	A	5	(1)	0	(6)	3	(2)	**22**
	B	5	(4)	4	(3)	5	(5)	
TOTAL		23		21		27		

5.2 Code comprehension

Code comprehension was established based on the participants' perception of being able to comprehend the code and on their ability to complete the task at hand with the help of a colleague. The information was gathered through a questionnaire answered by programmers at the end of each pair programming session. The survey used a seven-point Likert scale with anchors ranging from strongly disagree (1) to strongly agree (7).

Table 2 presents the results of the answers to five of the questions related to code comprehension. Although the programmers were in general able to understand the code, the responses to questions 1, 4, and 5, the ones more directly related to the understanding of the code, indicate that code comprehension was slightly better when the authors were collocated. A t-test comparing the two distributed conditions shows that there was no statistical difference between the two ($p < 0.05$).

Questions 3, 4 and 5 are related to the usefulness of the pair programming experience, which was in general rather high.

The subjects did not consider the programming tasks to be complex, but neither trivial (2.75 to 2.92).

We also asked them if they considered the help offered by their peer NOT useful. The responses were 1.8 (Collocated), 2.6 (NetMeeting), and 1.8 (COPPER). Thus, in general they considered the help provided as being useful, but less so in the NetMeeting condition.

Table 2. Perception of code comprehension

Question	Collocated	NetMeeting	COPPER
1. I understood the program	6	5.58	5.5
2. I found the program to be complex	2.75	2.92	2.75
3. Working with someone helped me find more defects	6.42	6.17	6.25
4. Working with someone helped me to modify the program	6.17	6	5.67
5. Working with someone helped me understand the program	6.33	6.08	5.83

With respect to the actual completion of the tasks assigned to the programmers, we have that 4 groups completed the modification when using NetMeeting, while 5 groups successfully modified the program using COPPER. These results give a slight advantage to the COPPER mode. However, the results from the questionnaires and the completion of the task DO NOT provide evidence that the awareness features incorporated in COPPER helped pairs understand the code more than NetMeeting did (H2 Not Accepted).

5.3 Observations on Actual Collaboration Support

Not being able to accept the second hypothesis (H2), an analysis of the actual collaboration awareness features that were included in COPPER to this end is required. Particularly, we focus on the features of i) Awareness of the Floor Control, ii) Concurrent Work and iii) Radar view. The results of this analysis can be summarized as follows:

- The usefulness of the awareness of floor control can't be appreciated when an additional (and simpler) coordination mechanism is available (in this case, a verbal spoken protocol)
- Taking advantage of the Radar view as a coordination mechanism requires experience on how to use it, which is not often the case for novice users of collaborative applications. In this case, the tool might hamper rather than support users in performing a task.
- Concurrent work support allows for independent browsing and editing of code, as well as awareness on the concurrent actions of the distributed pair, however, inexperienced programmers might end working in parallel, and thus not performing pair-programming.

A brief discussion on each of these points with supporting evidence from the study follows.

5.3.1 Awareness of Floor Control.
Table 3 shows the number of times each pair exchanged control of the floor in each phase of the programming exercises they conducted. It can be seen that in distributed mode the pair exchanged control of the floor almost three times more as when they were collocated. A t-test on the data shows that this difference is a statistically significant ($t = -2.2831$, $df = 5$, $p < 0.05$). In contrast the difference between the NetMeeting and COPPER conditions is not statistically significant ($t = -0.0674$, $df = 5$, $p < 0.05$). Apparently the mechanisms to exchange control of the floor provided by both collaborative tools (NetMeeting and COPPER) are simpler to use than moving the keyboard and/or rearranging chairs. Indeed, in the case of NetMeeting a user only needs to move the cursor or start typing to gain control of the floor, even tough in practice, as we observed in the videos, they always used verbal communication to either request or grant floor control. This also explains why the semaphore widget provided by COPPER to request control of the floor was seldom used. The pair would negotiate the exchange verbally, as they did when in Collocated or NetMeeting mode, and then simply press the button to grant control. The fact that the social protocol was effectively used for coordination is related to the perception by the programmers that there was a good communication between the pairs, supported by the use of audio over the telephone line. When asked to rate the assertion: "Communication with my pair was excellent", the results given were 6.5 (Collocated), 6.1 (NetMeeting), and 5.8 (COPPER). Even though the results for the collocated mode are higher, there's no statistical significance between the three conditions.

5.3.2 Concurrent Work and Radar View.
We noticed that when in COPPER mode, the observers took advantage of the opportunity of moving to different parts of the code. This was mostly the case in phases 2 (debugging) and 3 (modification). Actually, in the debugging phase one pair worked in parallel for a chunk of time, each looking for errors in different parts of the code (not actually doing pair programming). An interesting observation is that two couples experienced some difficulties using the radar view. One couple in particular spent a couple of minutes trying to find each other, finally deciding to meet at the top of the file. Apparently, users need to be more familiar with the radar view to take advantage of it. This might explain why COPPER users gave a lower grade (5.8) to the question "Coordination with my colleague was always good" when compared with the use of NetMeeting (6.1).

Table 3. Number of times pairs changed control of the floor in the pair programming sessions

Team	Collocated				NetMeeting				COPPER			
	Ph1	**Ph2**	**Ph3**	**T**	**Ph1**	**Ph2**	**Ph3**	**T**	**Ph1**	**Ph2**	**Ph3**	**T**
1	0	0	0	0	3	2	1	6	1	2	2	5
2	0	2	1	3	2	2	0	4	0	2	2	4
3	0	1	0	1	1	1	2	4	1	2	2	5
4	0	2	1	3	1	1	0	2	0	1	0	1
5	0	1	0	1	2	1	0	3	2	0	2	4
6	0	0	1	1	1	2	3	6	2	2	0	4
TOTAL	0	6	3	**9**	10	9	6	**25**	6	9	8	**23**

5.3.3 Concurrent Work and Awareness of Concurrent Actions.
We had also asked in the questionnaire if the pair programming experience was enjoyable, to which they answered with high marks: 6.2 (Collocated), 5.7 (NetMeeting), and 6.0 (COPPER). The lower grade given to NetMeeting could partially be explained by some difficulties experienced by NetMeeting users with the tool. In the questionnaire one student wrote "NetMeeting is very slow at refreshing the screen which creates errors", while another mentioned: "writing was slower than when only one person typed the code (in Collocated mode)". Analysis of the videos indeed show that at times the observer would get confused and think that the driver has stopped writing when in fact it was just his screen that would take some time to refresh. This problem was not observed with COPPER, which allows

concurrent work and provides better support for maintaining awareness of the concurrent actions of the distributed pair by updating the remote application one character at a time.

5.3.4 Evaluation Limitations.
It is important to highlight that our study was limited to simple programming tasks that could be completed in about one-hour each. This limitation was imposed by the need to control conditions of the experiment. In particular, we required the subjects to be completely involved in the task for the duration of the programming session (no additional parallel work) and the within-subjects experimental design required them to work on three different tasks. In addition, we believed that beyond three hours of work the concentration of the subjects on the task could be compromised. Since pair programmers are required to be continuously focused on the task, the experience can be intense and mentally exhausting (Williams and Kessler, 2000).

6 Conclusions and Future Work

Extreme programming techniques, and in particular pair programming, are gaining considerable attention given their advantage at handling software development projects with vague or changing requirements. As the software industry continues to grow and their practice becomes global, distributed teams will require appropriate tools to support their software development practices. The development of such tools needs to be supported by empirical research aimed at establishing the necessary services required to support the intensive collaboration required during this practice.

Towards this end we have designed, developed and evaluated COPPER, a collaborative environment to provide support for distributed pair programming. Concerning its design and implementation, COPPER is successfully based on the concepts of Actual and Potential Collaboration Spaces and Awareness, which facilitate the separation of concerns regarding the provision of support for i) establishing an opportunistic distributed pair programming session, and ii) actually performing pair programming in a distributed manner. Concerning its evaluation, on the one hand, our results seem to support our hypothesis that distributed groups could be as effective in finding programming errors as collocated ones. On the other hand, the additional awareness features provided by the COPPER tool didn't seem to facilitate the programmer's understanding of the code over what simple application sharing can accomplish when voice support is also available.

Overall, according to evaluation results and comments from evaluation participants', COPPER improves Distributed Pair Programming, in subtle, but significant ways, such as allowing concurrent work and maintaining awareness on the concurrent actions of the distributed pair, in contrast to collaboration-transparent applications (e.g. NetMeeting) which do not provide these features. However, actually taking advantage of COPPER's more advanced features (e.g. Awareness of the Floor Control and Radar View) require users to be more experienced in the use of these collaboration-aware features.

Finally, in this study we did not evaluate the use of COPPER support for potential collaboration (user and document presence tool). This requires monitoring the programmers for longer periods of time. This way, future work involves performing additional research i) to evaluate the usefulness of the support for potential collaboration awareness, as well as, ii) to evaluate whether COPPER support for actual collaboration performs better when using other coordination mechanism rather than voice communication.

References

1. **Abrahamsson, P., Warsta, J., Siponen, M.T., Ronkainen, J.**, New directions on agile methods: a comparative analysis, Proceedings of the 25th International Conference on Software Engineering, May 03-10, 2003, Portland, Oregon, pp. 244-254.
2. **Baheti, P., Gehringer, E., and D. Stotts**. Exploring the efficacy of distributed pair programming, Proceedings Extreme Programming and Agile Methods - XP/Agile Universe 2002, August 2002, p. 208-220.
3. **Beck, K**., Extreme Programming Explained, Embrace Change, Addison-Wesley, pp. 190, 2000.
4. **Bly S., Harrison S. and Irwin S**., "Media Spaces: Bringing People Together in a Video, Audio and Computing

Environment", Communications of the ACM, 36(1), January 1993.
5. **Calvary, G., Coutaz, J., and Nigay, L**: From Single-User Architectural Design to PAC*: a Generic Software Architecture Model for CSCW, Proceedings of CHI 97, ACM publ., pp. 242-249.
6. **Cockburn A. and Williams L.**, The Cost and Benefits of Pair Programming. Addison Wesley. (2001).
7. **Ellis, C.A., Gibbs, S.J., and Rein, G.L.**, Groupware: Some issues and experiences, CACM, Vol. 34, No. 1, pp. 38-58.
8. **Gaver, W., Moran, T., Maclean, A., Lövstrand, L., Dourish, P., Carter, K., Buxton, W.**: Realizing a Video Environment: EuroPARC's RAVE System. In: Proc. CHI'92, ACM Press, Monterrey CA (USA). May 3-7 (1992) 27-35.
9. **Herbsleb, J.D. and Mockus, A.** (2003). An Empirical Study of Speed and Communication in Globally-Distributed Software Development. IEEE Transactions on Software Engineering, 29(3), 2003, pp. 1-14.
10. **Herbsleb, J.D. and Moitra, D.**, Global Software Development. IEEE Software. 18(2), (2001), 16-20.
11. **Kircher M., Jain P., Corsaro A., and Levine D.**, Distributed Extreme Programming. Extreme Programming and Flexible Processes in Software Engineering, Italy, May, (2001)
12. **Morán, A. L., Favela, J., Martínez, A. M. and Decouchant, D.**, Document Presence Notification Services for Collaborative Writing. In Proc. of CRIWG'2001. IEEE Computer Press. Darmstadt, Germany, Sept. 6-8, (2001), 125-133.
13. **Morán, A. L., Favela, J., Martínez, A. M. and Decouchant, D.**, On The Design of Potential Collaboration Spaces, in "Current Approaches for Groupware Design, Implementation and Evaluation", Borges, M., Haake, J. and Pino, J. (Eds.), The International Journal of Computer Applications in Technology (IJCAT), ISSN 01952-8091, Vol. 19, No. 3/4, 2004, pp. 184-194.
14. **Natsu, H., Favela, J., Morán, A.L., Decouchant, D., and Martinez, A.M.**, Distributed Pair Programming in the Web. In Proc. ENC'03, IEEE Comp Society, Mexico, 2003, 81-88.
15. **Sanders, D.**, "Student Perceptions of the Suitability of Extreme and Pair Programming", in Extreme Programming Examined M. Marschesi, and G. Succi, D., Wells and L. Williams (eds.) Boston, MA, AddisonpWesley, 2002, pp. 261-271.
16. **Stephens, M. and Rosenberg, D.**, Extreme Programming Refactored: The Case Against XP, Apress, pp. 432, 2003.
17. **Stotts, D., Williams, L., Nagappan, N., Baheti, P., Jen, D. and A. Jackson**, Virtual Teaming: Experiments and experiences with distributed pair programming, Proceedings of the Third XP Agile Universe Conference (Springer LNCS 2753), pages 129 - 141, August 2003.
18. **Tang, J. C., Isaacs, E. A., Rua, M.**: Supporting Distributed Groups with a Montage of Lightweight Interactions. In: Proc. of CSCW'94, ACM Press, Chapel Hill NC (USA) October 22-26 (1994) 23-34.
19. **Warsta, J. and Abrahamsson, P.**, Is Open Source Development Essentially an Agile Method?, 3rd Workshop on Open Source Software Engineering, Portland, Oregon, 2003.
20. **Whitehead E. J.**, Collaborative Authoring on the Web: Introducing WebDAV. Bulletin of the American Society for Information Science, 25(1), (1998), 25-29
21. **Williams L. and Kessler R.**, All I Really Need to Know about Pair Programming I Learned in Kindergarten. CACM, 43(5), (2000), 109-114.
22. **Williams L., Kessler R.**, Cunningham W., Jeffries R., Strengthening the Case for Pair Programming. IEEE Software, 17(4), (2000), 19-25

Alberto L. Morán *is a Professor of Computer Sciences at the Universidad Autónoma de Baja California (UABC) in México, where he leads the Intelligent Environments Group. His research interests include distributed software development, informal collaboration, CSCW, and ubiquitous computing. He holds a B.Sc. in Computer Sciences from UABC; a M.Sc. in Computer Sciences from CICESE, and a Ph.D. in Computer Sciences from the Institut National Polytechnique de Grenoble (INPG), France. He is a Member of the ACM.*

Jesús Favela *is a professor of computer science at CICESE, where he leads the Mobile and Ubiquitous Healthcare Laboratory and heads the Department of Computer Science. His research interests include ubiquitous computing, medical informatics and CSCW. He holds a BS from the Universidad Nacional Autónoma de Mexico (UNAM) and MSc and PhD from the Massachusetts Institute of Technology (MIT). He is a member of the ACM and the American Medical Informatics Association (AMIA), and former president of the Sociedad Mexicana de Ciencia de la Computación (SMCC).*

Raúl Romero Wells *is a lecturer in the Computer Science Department at CICESE. His interests include databases, experimental design, and medical informatics. He holds a BSc from the Universidad Autónoma de Baja California (UABC) and a MBA from Centro de Enseñanza Técnica y Superior (CETYS) in México.*

Hiroshi Natsu *is a software engineer at Code Services, a software development and maintenance company located in Ensenada, México. His research interests include distributed software development and CSCW. He holds a M.Sc in Computer Science from CICESE and a BS in Computer Engineering from the Universidad Autónoma de Baja California (UABC) in Mexico.*

Cynthia Pérez Castro *received a B.Sc. degree from Universidad Autonoma de Sinaloa, in 2002. In 2004, she obtained the M.Sc. in computer science from CICESE, Mexico, where she is currently working towards the Ph.D. in Computer Science. Her research interests include human-computer interaction, evolutionary computation, machine learning and computer vision. She is a student member of the IEEE Computer Society.*

Omar Robles *is Process Manager at Code Services, a software development and maintenance company. His research and work interests include Software Quality and Software Process Improvements. He holds a M.Sc in computer science from CICESE and a B.S in Informatics and Statistics from the Universidad Autónoma de Nayarit (UAN) in Mexico.*

Ana María Martínez Enríquez *received the Master and Ph.D degrees in Computer Science from the University of Paris VI (France) in 1982 and 1985 respectively. After her PhD, she focused on artificial intelligence, natural language processing, machine learning and design and implementation of knowledge based systems. Nowadays, her research interests cover two complementary domains: Computer Supported Cooperative Work and Distributed Artificial Intelligence. Currently, she is a Research Professor of the Computer Science Department at the Mexican Center for Scientific Research and Advanced Studies at the National Polytechnic Institute (CINVESTAV-IPN).*