Tool Support for Distributed Pair Programming

Brian F. Hanks
University of California, Santa Cruz
brianh@soe.ucsc.edu

June 17, 2002

Geographically distributed project teams and organizations are becoming more common. For example, in many organizations workers from multiple locations collaborate on projects. Telecommuting is becoming more common, yet these workers are also project team members. Open source software development is an extreme example of this movement, since there are typically no collocated developers at all.

Conflicts between the desire to use pair programming and factors motivating distributed teams may result in organizational change. For example, Morales [1] reports that introducing pair programming at one organization resulted in reduced telecommuting due to the need to be in the office.

Although pair programming has not been studied extensively, early research indicates that it results in software with fewer errors. While the total person effort may be slightly larger [2] than that expended by traditional programming methods, the elapsed time is reduced. Additional benefits of pairing include higher confidence in solutions [3], improved staff morale and increased job satisfaction [4].

At UC Santa Cruz, pair programming has been successfully used as a teaching tool in introductory programming courses [5, 6]. Unfortunately, some students have had difficulty participating due to the collocation requirement, largely due to scheduling conflicts and personal desires not to meet in campus labs at night.

Although some efforts have been made to apply pair programming or XP in a distributed manner [7, 8], little quantitative evidence of its success has been published. Anecdotal evidence of the potential of distributed pair programming can be found on the WikiWikiWeb [9] and on the comp.software.extreme-programming newsgroup. These sites indicate that distributed pair programming is being attempted by actual practitioners with varying degrees of success.

These efforts have used off-the-shelf application sharing tools such as Microsoft's NetMeeting in their attempts to support distributed pairing. While these tools may be excellent at supporting distributed meetings, they are not specifically designed for pair programming. As such, their notions of floor control and telepointing are not appropriate for distributed pairing. Additionally, these tools do not typically support heterogeneous computing environments. This limits their applicability and may prevent them from being used in many potential distributed pairing situations.

I believe that the benefits of pair programming can be extended to non-collocated pairs by providing appropriate collaborative software to support them. In industry, this would allow members of distributed teams to pair program. In academia, this would provide more opportunities to pair, thus reducing scheduling conflicts and enabling the use of pairing for remote students. Finally, if students could pair from their home or dorm, they would be able to pair at night without fear.

CSCW research on collaborative tools has taken two approaches to support synchronous activities: shared desktops and collaboration-aware tools. Ellis [10] discusses the tradeoffs between these approaches. The shared desktop approach replicates one user's desktop onto multiple workstations so that all users see and interact with the same view. This approach allows single-user applications to be shared by multiple users without modification. Collaboration-aware tools must be specifically developed, but these tools can provide a richer user interface that more effectively supports collaboration.

One approach to supporting distributed pair programming is to modify a set of programming tools so that they are collaboration-aware. This would require a large effort and would not be generally applicable due to the wide variety of tools, languages, and development environments used. For example, a collaboration-aware Java development environment would only be useful to Java developers, and even among them it would face acceptance difficulties due to each developer having personal preferences in their choice of tools. I believe that a shared desktop approach will be more generally applicable, as it will provide programmers the ability to pair while still using their favorite tools.

It is unclear whether a distributed pairing tool needs to support video. Watching collocated pair programmers, I have noted that they spend the majority of their time focused on the screen, even while they are speaking with each other. At this time, I believe that an audio connection is sufficient to support distributed pairing. This can be implemented using either a separate telephone connection, or by using a chat tool such as AOL Instant Messenger.

An additional downside of video is the bandwidth required to support a reasonable resolution. In some situations, such as home telecommuting, video may not even be feasible.

I am enhancing the Virtual Network Computing (VNC) [11] application to support distributed pairing. VNC is an open source desktop sharing tool developed at AT&T Laboratories Cambridge. While AT&T researchers have primarily used this tool to support ubiquitous computing, it has many of the features required to support distributed pair programming.

In the standard version of VNC, multiple clients can connect to a VNC server. However, keyboard and mouse events from the separate clients are interlaced into a single stream that is sent to the underlying application. Although pairs could use VNC this way, this behavior is not optimal. For example, if the navigator wishes to point to an area on the screen by using the mouse pointer, the driver has to refrain from using her mouse. Additionally, the navigator has to make certain that she doesn't accidentally press any keys or use any mouse buttons while the driver is active.

I am working on modifying VNC to support distributed pairing. These modifications include the addition of telepointing to allow gesturing by the navigator. I am also adding logging of the pairing process, which will provide more accurate records of the amount of time spent pairing, driving and navigating by each partner. Collecting accurate data in pair programming experiments has been challenging [6], particularly when the data is self-reported, and this tool will address that problem.

At this time, floor control will be maintained using social means. With social floor control, a CSCW application does not impose floor control technically, but instead lets the users develop their own social mechanisms for controlling the floor. In this configuration, any user can type at any time, and the application merges the input streams. Although social floor control can be chaotic, Greenberg [12] found that it can be effective in groups of two or three people. I believe that this is be the best approach in a distributed pair programming tool. Since collocated pairs tend to casually change roles, an application mediated floor control mechanism may actually hinder the pair programming process.

I plan to use this tool to study distributed pair programming, as I believe that distributed pairing can be as effective as collocated pairing. I hope to use this tool to study pairs in introductory programming classes, with some students using the tool and others following the traditional pairing model.

I believe that pair programming is a significant software development technique that can lead to higher software product quality. Unfortunately, its limited applicability due to its collocation requirements is hindering its adoption. It is essential to have simple, portable, intuitive tools available to support effective distributed pairing.

References

[1] Alexandra Morales. Going to extremes. *Software Development*, 10(1), January 2002. `http://www.sdmagazine.com/print/documentID=20202`.

[2] Laurie A. Williams. *The Collaborative Software Process*. PhD thesis, University of Utah, 2000.

[3] Laurie A. Williams. Strengthening the case for pair programming. *IEEE Software*, 17(4):19–25, July/August 2000.

[4] John T. Nosek. The case for collaborative programming. *Communications of the ACM*, 41(3):105–108, March 1998.

[5] Charlie McDowell, Heather Bullock, Julian Fernald, and Linda Werner. A study of pair-programming in an introductory programming course. In *Proceedings*

of the 33rd ACM Technical Symposium on Computer Science Education, page TBD, February 27 - March 3, 2002.

[6] Jennifer Bevan, Linda Werner, and Charlie McDowell. Guidelines for the use of pair programming in a freshman programming class. In *Proceedings of the 15th Conference on Software Engineering Education and Training*, pages 100–108, February 25-27, 2002.

[7] Michael Kircher, Prashant Jain, Angelo Corsaro, and David Levine. Distributed eXtreme Programming. In *Proceedings of XP2001*, May 2001. `http://www.xp2001.org/xp2001/conference/papers/Chapter16-Kircher+alii.pdf`, current May 29, 2002.

[8] Frank Maurer and Sebastien Martel. Process support for distributed extreme programming teams, 2001. `http://sern.ucalgary.ca/~milos/papers/2002/MaurerMartel2002.pdf`, current May 29, 2002.

[9] WikiWikiWeb. `http://c2.com/cgi/wiki?WikiWikiWeb`. See virtual pair programming discussion at `http://c2.com/cgi/wiki?VirtualPairProgramming`.

[10] C.A. Ellis, S.J. Gibbs, and G.L. Rein. Groupware: Some issues and experiences. *Communications of the ACM*, pages 39–58, January 1991.

[11] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, January-February 1998.

[12] Saul Greenberg. Sharing views and interactions with single-user applications. *ACM SIGOIS Bulletin*, 11(2-3):227–237, April 1990.