Another Person's Eye Gaze as a Cue in Solving Programming Problems

Randy Stein
Department of Computer Science
State University of New York
Stony Brook, NY 11794-4400
rbstein@optonline.net

Susan E. Brennan
Department of Psychology
State University of New York
Stony Brook, NY 11794-2500
susan.brennan@sunysb.edu

ABSTRACT

Expertise in computer programming can often be difficult to transfer verbally. Moreover, technical training and communication occur more and more between people who are located at a distance. We tested the hypothesis that seeing one person's visual focus of attention (represented as an eyegaze cursor) while debugging software (displayed as text on a screen) can be helpful to another person doing the same task. In an experiment, a group of professional programmers searched for bugs in small Java programs while wearing an unobtrusive head-mounted eye tracker. Later, a second set of programmers searched for bugs in the same programs. For half of the bugs, the second set of programmers first viewed a recording of an eyegaze cursor from one of the first programmers displayed over the (indistinct) screen of code, and for the other half they did not. The second set of programmers found the bugs more quickly after viewing the eye gaze of the first programmers, suggesting that another person's eye gaze, produced instrumentally (as opposed to intentionally, like pointing with a mouse), can be a useful cue in problem solving. This finding supports the potential of eye gaze as a valuable cue for collaborative interaction in a visuo-spatial task conducted at a distance.

Categories and Subject Descriptors

H.5.3 [Information Interfaces and Presentation]: Group and Organization Interfaces - *collaborative computing asynchronous interaction, synchronous interaction, computer-supported cooperative work.* H.4.3 [Information Interfaces and Presentation]: Communications Applications - *videoconferencing*.

General Terms

Human Factors, Experimentation.

Keywords

Eye tracking, visual co-presence, mediated communication, gaze-based & attentional interfaces, programming, debugging.

1. INTRODUCTION

Achieving a joint focus of attention is critical for successful communication and collaboration. While speaking, people provide many cues about their focus of attention with the syntactic and lexical choices they make, as well as with intonation; such information can be represented in a context model that enables addressees to interpret referring expressions that might otherwise be ambiguous (see, e.g., [5] [12] [13]). For a physical task, particularly one with a visuo-spatial component, being able to see what a partner is doing makes communication much more efficient than not seeing such information ([3] [4] [6] [8] [22]). When collaborators communicate at a distance, the most useful kind of visual information appears to be a shared task artifact; studies of video conferencing have shown that being able to view a document that a conversational partner is looking at during a joint task is more useful than being able to see the partner's face ([25] [27] [34] [35]).

Unfortunately, when people collaborate remotely (whether at different places or different times), it can be difficult for them to become aware of one another's focus of attention [7]. A number of technologies have been invented to support joint activity and awareness; for instance, GroupWeb [15] slaves web browsers together so that when one is scrolled, the other follows, and Piazza [19] enables people to see (and contact) others who are visiting the same web site at the same time. Other systems enable experts to visualize and control novices' workstations remotely, in order for the experts to help debug problems; these are more effective than telephone help alone. Some of these systems display where a partner's input cursor is, enabling deixis (referring by pointing). And many applications permit people working on a file at different times to intentionally highlight changes; the problem of version control, for both software and text documents, can be ultimately construed as a problem of tracking joint activity. Researchers have explored the possibility of representing presence using virtual embodiments that can point to screen objects [18].

However, none of these technologies provides the precise, moment-by-moment information about a collaborator's focus of attention that, we hypothesize, is available in *eye gaze*. People are very sensitive to their interlocutors' gaze and eye movements in certain situations; consider, for instance, how difficult it is to glance surreptitiously at one's watch during a face-to-face conversation. Eye gaze can provide multiple cues; it can point or select, serve as a general display of attentiveness (or conversely, of distraction), support attributions about ongoing cognitive activity such as searching, comparing two objects, or deliberating, or give evidence about the status of a physical task. A glance may be not only *instrumental* (to the extent that looking is an unavoidable part of performing a visuo-spatial task), but also *informative* (observers may be able to use partners' eye gaze in interpreting an utterance or recognizing an intention), or even *communicative* in the sense described by Grice [11] in that the gazer may intend for the observer to see where she is looking and to recognize that she intends to point or display attention to an object. In video conferencing, it has been documented that *gaze awareness* ([9][20][32]) can enable one person to estimate where anotherís focus of attention is in a shared workspace.

Eye gaze provides incremental and detailed information that is missing from more intentional forms of pointing with a mouse, trackball, touchpad, or stylus (see [1][2][4]). For instance, in a puzzle task in which individuals assembled a copy of a geometric model using a set of colored square pieces, patterns of eye fixations to the model showed that people encoded location and color in separate steps; this would not have been apparent from observing the mouse movements alone ([1][2]). In this situation, eye movements were informative about the encoding processes that preceded the decision about where to move the mouse.

On the other hand, eye movements are generally more ambiguous than mouse movements, simply *because* they are under less intentional control; so eye gaze is a poor choice of input modality for controlling a computer application with explicit commands when a user has other reliable input modalities available ([21]). Further, in most tasks there are multiple attributions for a given pattern of gaze (akin to the mode problem in HCI). Because of such issues, Jacob [21] has advocated using eye movements for *implicit* rather than explicit commands in human-computer interaction.

We agree with this assessment. The work we report here is part of a larger project on eye movements in mediated communication, in which we are using eye movements just as they are used in human communication--as implicit indicators of visual attention that can be used to achieve a joint focus of intention between two people collaborating remotely or asynchronously. But rather than having to estimate where a person is looking as in face-to-face communication, one person sees another's eyegaze represented as a cursor moving over a screen. We set out to demonstrate that having a visual representation of one person's eye gaze can improve another's performance in a visuo-spatial domain, namely, debugging computer programs.

2. DEBUGGING COMPUTER CODE: A VISUO-SPATIAL TASK

Beginning programmers often turn to experts for help with a programming problem, only to find that the experts have trouble articulating what they know. Experts may be able to find bugs quickly without being able to teach beginners how to do so on their own because expertise involves pattern-recognition that occurs without explicit awareness [28]. Thus it is worth investigating non-verbal ways that experts might use to communicate and transfer their knowledge to novices.

Past research on the differences between expert and novice programmers has been focused partially on the types of bugs made more commonly by novices. A few major categories of bugs tend to account for most novice errors. However, most novice bugs are not based on improper use of programming language constructs as many have assumed, but rather are a result of complicating or improperly carrying out programming goals. A goal can be any task that a program is supposed to carry out, such as input validation or a calculation. Novice programmers tend to make errors when they try to merge goals, as one of the goals is often seen as less important and gets carried out incorrectly or ignored entirely. Construct-based errors, such as using logic operators incorrectly (i.e., using an ìorî operator where an ìandî should be) are significantly less common but still arise ([31][29][30]).

Other research has focused on the different ways in which experts and novices comprehend code. Expertsí mental representations tend to be more abstract and contain more inter-connected layers than novicesí representations. Experts tend to recognize common programming patterns more and are more able to remember specific parts of the code [10]. They tend to build a top-down representation of a program and focus only on relevant information needed to solve a problem [23]. When given a problem, along with documentation and the code itself to solve it, experts tended to rely on the documentation to get a sense of where the relevant code was and needed to look at only 20% of the code.

Research focusing specifically on debugging has shown that programmers tend to use *slicing* while debugging. Slicing refers to mentally splitting up a program into sets of lines that are related by a common data flow (i.e. referencing the same data). Lines of code that do not affect those data are effectively stripped from that part of the programmerís mental representation of the program. Programmers build these mental slices as they work backwards through a program, starting with the source of the bug [33]. In a study focusing specifically on the differences between novice and expert debugging [14], experts found more bugs, found them faster than novices, and tended to spend more time building a mental representation of the program and on program comprehension. Experts also tended to be quicker to test potential solutions. It is also worth noting that although this study also kept track of what the programmers were paying attention to (although not with an eye tracker), it tracked only whether the subject was reading code or reading a problem description. There was no difference in the way experts and novices divided their time between these two activities [14].

Because the domain of software debugging is of broad interest to many in the HCI and CSCW communities, and because it involves a screen-based visuo-spatial task on which people often collaborate, we chose this domain in which to investigate the utility of eye gaze in problem-solving. Although we are ultimately interested in realistic, real-time, collaboration between remotely located, intracting partners with different perspectives or knowledge (such as experts and novices), for this initial study we made some simplifying assumptions aimed at demonstrating the utility of viewing another person's instrumentally-produced eye gaze. We had expert programmers view the pre-recorded eye gaze traces of other experts (who were not intending to communicate anything deictically about where they were looking) to see if awareness related to specific bugs could be transferred, enabling the viewers to find bugs faster. While it seems likely that seeing a pointer to the part of a text file that needs to be changed in order to fix the bug (the edit point) would be helpful by itself, we reasoned that information in the visual search *leading up to* the edit point would also be useful, and in fact might be essential for viewers not only to find the edit point, but also to know what needed to be done to fix the bug. On the other hand, it is possible that watching someone else's search pattern could slow down a viewer's attempt to find the bug, if the gaze trace were ambiguous or distracting. In the domain of software debugging, just what information is available in a collaborator's instrumental eyegaze?

3. EXPERIMENT

3.1 Design

The experiment was divided into two phases. In the first phase, four professional programmers visited the lab individually and found bugs in three small Java programs while wearing a lightweight (6.5 oz.) head-mounted ISCAN RK-726PCI pupil/corneal reflection eye tracker. A miniature camera mounted on the visor recorded the programmer's view of the screen, and another camera tracked the programmer's eye gaze, which was overlaid over the screen image; this video image was recorded. The programmers were asked to think out loud while finding the bugs, and their speech was recorded. In the videotaped recordings, the screens of code were not legible; the only information available was the pattern of the eye gaze trace displayed over a spatially correspondent screen layout where the lines of code were blurred.

In the second phase, six different professional programmers found bugs in the three programs viewed by the original group. However, for half of the bugs, they first viewed a (silent) videotape of another programmer's eye gaze trace for that bug; the rest of the time, they searched for the bug without having seen the eye gaze trace. Programmers in Phase 2 did not wear the eye tracker, and they were not asked to think out loud.

A total of eight bugs from the first phase were used as stimuli in second phase. Two stimulus videotapes were made containing four bugs each, and three participants in the second phase were randomly assigned to view each tape. This design enabled us to examine the effect of eye gaze cues within-subjects and within-items. The bugs were blocked by visual evidence; half the subjects solved four bugs with visual evidence and then four without; the other half of the subjects solved the bugs in the same order but did the first four without visual evidence and the second four, with. So the experimental design varied eyegaze information both between- and within-subjects.

3.2 Stimuli

The stimuli came from three Java programs designed to be similar to real programs a novice programmer might be asked to debug. They were selected so that all parts of the program could fit on one screen, as scrolling would have made it difficult for programmers in the second phase to map the videotaped eye gaze trace onto the screenfull of code.[1] In addition, the text had to be large enough so that looking at the cursor in the eye gaze video could reasonably indicate which line within the program's layout a subject was looking at. The programs were based on real assignments given to students in Stony Brookís introductory programming course (CSE 114) or were simple ìtoyî programs built to demonstrate a single concept or algorithm.

The bugs themselves were chosen to emulate common errors a novice programmer might make. In addition to basic errors with programming language constructs (such as using an incorrect operator), they included bugs caused by the following: combining two operations into one, improper understanding of an algorithm, incorrect conclusions about a problem (e.g., a value is calculated improperly), and plan-based errors, in which the wrong common programming ìtemplateî is used to solve a problem. They were chosen to represent a range of difficulty and several degrees of modularity, meaning the extent to which the program was broken up into different functions as opposed to one large function.

The three Java programs were:

1. Palindrome, a simple program that tested whether or not a sequence of characters is a palindrome. Errors in this program included a problem with logical programming constructs and two more bugs caused by trying to do complex manipulations of strings in a single line where it would be easier to break it up between two or more lines. Incorrect understanding of Java strings could also cause the bugs. Palindrome contained a moderate degree of modularity, as it was broken up into three functions within a single file. Two of the bugs were close to each other near the top of the screen while the third was towards the bottom.
2. Shopper, a shopping cart simulator that allowed users to interact with a simulated shopping cart. This program contained one construct-based and one plan-based bug. It was also split into many functions spread across three files and was the most modular of the stimuli (note that the eye gaze videos were not complicated by the fact that the program had three files because the subjects could only look at one file at once). Both bugs were in the same file, with one near the middle and one at the bottom.
3. Triangle, a triangle drawing program. The three errors were based around incorrect geometric calculations. Triangle contained only one function and thus had the lowest degree of modularity. One bug was near the top of the program, one was in the middle, and the last was towards the end.

Note that all the bugs were logical rather than syntactic, meaning that all the programs could run; they just did not run correctly.

Out of 32 potential stimulus videos recorded in Phase 1, eight videos for Phase 2 (one per bug) were chosen based on the criteria of success (the programmer had to have found the bug), accuracy (the eye gaze trace had to accurately represent where the subject was actually looking, within 1-2 lines), length (shorter

[1] In a more realistic version of this debugging task, the screens of the collaborators would need to scroll in synchrony in order for them to share a focus of visual attention.

durations were preferred to longer ones, to minimize the memory load on programmers in Phase 2), typicality (based on the first author's intuitions as a Java programmer), and insightfulness (traces that showed the subject looking through several portions of the code and referring to related areas before zeroing in on the bug location). Tape A contained videos of subjects finding all three of the bugs for Palindrome and one of the bugs for Shopper. Tape B contained videos of the other bug for Shopper and all three bugs for Triangle.

3.3 Participants

The four programmers in Phase 1 who generated the stimuli and the six who viewed them in Phase 2 were recent computer science graduates (within one year) from a variety of schools, now working professionally as software engineers. They were all experts relative to students just starting an undergraduate computer science program. They each received $7 for participating in the study. The programmers in Phase 1 were male, as were five out of six in Phase 2.

3.4 Procedure

Programmers in the first phase wore the head-mounted eye tracker. After a brief calibration, the experimenter (the first author) gave each programmer an overview of what each program was supposed to do, a demonstration of the properly working program, and a demonstration of the incorrect version that needed debugging. Then the experimenter displayed the code onscreen and provided any necessary explanation of the object classes in the program. Finally, the programmer was given up to ten minutes to read through the program and find as many bugs as possible. Programmers were provided with scrap paper to help with any calculations they might have to do; however, they did not use the scrap paper much, so the eye gaze videos did not contain much off-screen looking. They were also allowed to ask questions about the code as needed. Programmers were encouraged to think out loud as they searched, so that the experimenter could tell that they were finding the bugs correctly. Average times to find the eight bugs ranged from 81 to 225 seconds. Each time a programmer found a bug, he was instructed to say ìI found a bugî and explain what the error was and what the solution was (the programmers did not actually change the code and test their solutions). The programmer was then asked if he wanted to continue. After he wanted to stop or the ten minute period was up, he was informed about which bugs he had found correctly and which he had missed.

At the outset of Phase 2, the programmers in the second group were told that they would be seeing videos of another programmer finding some, but not all, of the bugs that they themselves would have to find. They were also told that their performance would be timed. The procedure in Phase 2 was otherwise identical to that in Phase 1, except that in the block of bug-finding trials with visual evidence, the programmer either viewed the videotape of the eye gaze trace for the first bug in that program just before seeing the screenfull of code, or not. Because the actual words in the code were indistinct on the videotape, he or she then had to remember the spatial information about where the bug and related lines were in the code layout. The eight videotaped searches used as stimuli in Phase 2 ranged from about 30 seconds in length to about 2.5 minutes. When a program (either Palindrome or Triangle) had multiple bugs in the visual evidence condition, the programmer alternated between viewing the video and trying to find the corresponding bug for up to six minutes. For Shopper, the programmer viewed whichever video he or she was assigned to, tried to find that bug, and then tried to find the other bug without first seeing a video. In the block of bug-finding trials without visual evidence (either Triangle or Palindrome), programmers had ten minutes to find as many bugs as possible, just as in Phase 1. They could abandon the search for a bug at any time.

After showing a programmer a video, the experimenter made sure that he or she had paid attention to it and offered a repeat viewing, since memory load was not being tested here. Then the programmer was shown the legible code on the screen. For each bug, the time from when the programmer began searching to when he or she said ìI found a bugî was recorded. After identifying the bug to the experimenter, the programmer was told whether it was correct. Timing was done using a stopwatch.

3.5 Results

The 3 trials in Phase 2 in which participants did not find bugs were not included for analysis. Another 3 trials that took longer than 2 standard deviations above the mean (over 300 seconds) were discarded.[2]

As we hypothesized, participants in Phase 2 were faster to find a bug after viewing an eye gaze trace than without this visual cue. Viewing an eyegaze trace provided an advantage of about 62 seconds, on average. In an ANOVA with the data averaged by-subjects, F(1,5) = 6.17, p = .056; with the data averaged by-items, F(1,7) = 19.02, p = .003 (see Figure 1).

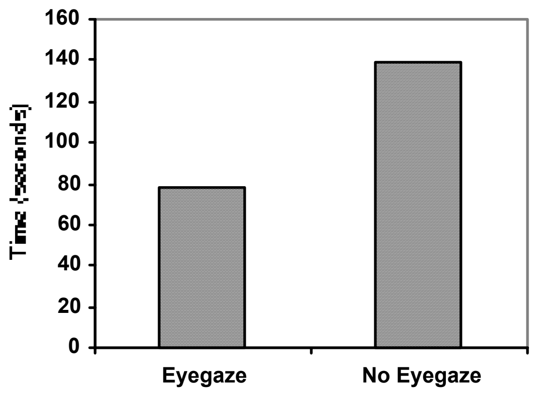

Figure 1. Mean bug-finding times, with and without eyegaze

[2] Bugs B and C in Palindrome were located rather close together; if a programmer found bug C after seeing the video for B, the time was included under bug C. Trials in Shopper where the subjects saw a video for bug E but found bug D first were included under the no gaze condition for bug D, since the subjects knew that D was not the intended bug and went on to continue to search for bug E. Bug D was one of the easiest bugs and immediately popped out to some who glanced at it while searching for information relevant to the whole program.

The only bug that programmers took longer on average to find in the eye gaze condition than in the no gaze condition was bug G in the Triangle problem. One participant took over five minutes to find this bug; upon debriefing, he explained that he had trouble remembering where the eye gaze in the video ended, as the movement of the eye gaze cursor was relatively erratic during the closing moments of the video. Another participant who took over two and a half minutes to find this bug echoed that comment. This suggests two things: (1) that the advantage of seeing an eye gaze trace before seeing the code is limited by having to remember it and use it later (a limitation in this pilot method that would not be present in a real-time version of the task), and (2) that eye gaze can be confusing if it does not follow a simple path (an effect that may be present in a real-time version of the task).

Only one participant was slower to solve problems for which he was in the eye gaze condition as opposed to the no gaze condition. This participant found two of the three bugs in the Palindrome program (for which he was in the no gaze condition) unusually quickly (in less than one minute). (Overall, the bugs in Palindrome were easier to find than those in other two programs.) On the Triangle program in the eye gaze condition, this participant took over two and a half minutes to find bug F (which, as previously explained, had a somewhat confusing video) and over four minutes to find bug G. During debriefing he explained that he was having trouble checking the calculations involved in drawing the triangle. Thus, although the deictic cues provided by eye gaze steered him in the right direction, they still could not compensate for his incomplete understanding of the problem.

Two participants could not find bug E at all; one was in the eye gaze condition, and one was in the no gaze condition. In both cases, the subjects knew with near-certainty where the bug was but could not pinpoint *what* it was. This confusion was due to the fact that the bug was designed as a common programming plan that ìlooked rightî, when in fact the wrong plan was being implemented. Both participants were misled by the plan, but perhaps more importantly, even though the one in the eyegaze condition knew where the bug was he could not overcome his disposition towards believing that the plan was correct and thus could not find the bug.

For one of the bugs, bug B on the Palindrome problem, the eyegaze video sped up the subjects who saw it, but not as much as expected. The bug was designed to be relatively simple ñ a long if-statement with its *and*s and *or*s reversed. We thought the error would be immediately apparent just from the beginning of the statement (which tested for a value less than 60 but greater than 90, clearly impossible). However, even after seeing the eye gaze of another programmer and seeing which line the bug was on, participants still took substantial time to figure out exactly what the error was. They all expressed that they had trouble because the statement was relatively long and did not contain any simplifying parenthesis, and so it was difficult to mentally parse the statement to find the error. This problem might also have been due to the eyegaze video itself, though; it was relatively long (two and a half minutes, the longest of any video) and showed eye gaze focusing on the problematic line for most of the time rather than looking around. After seeing the video, one participant remarked ìthis must be hard.î Thus, in this instance the eyegaze video might have made the bug look harder to find than it actually was.

When participants were solving Shopper, the largest and most modular of the three programs, the ones who had seen an eyegaze trace tended to go straight to the section containing the bugs after considering the other files either very briefly or not at all. Thus, for larger programs, an eyegaze cue may significantly cut down the amount of code that needs inspecting.

3.6 Discussion

Eyegaze information may have helped people find bugs in two ways: by pointing out where in the code the edit point for the bug was, and by circumscribing the slice of information related to the bug during the visual search process. An observer can reasonably assume that if the eyegaze cursor consistently moves back and forth between where the bug is and other sections of the code, then the other sections are related to the bug in some way. For the simplest bugs in our stimuli, the actual ending of the eyegaze videos may have been the most useful piece of information to the subjects. We expect that the logical information that may be derived from the full tracing of a programmerís search would probably be more useful, the more complex the bug, and the longer the program.

It is perhaps surprising that we found a clear effect of eye gaze with so few participants and debugging problems, especially given the other limitations placed upon the participants. Programmers in Phase 2 had to rely on memory while mapping the cues from the videotape of each eyegaze trace to its subsequent screenfull of code. If they could have viewed the eyegaze trace superimposed over legible code, the cues provided by eye gaze might have been even more useful. We expect further advantages to having gaze cues with the addition of a voice link and the ability to interact with a temporally co-present collaborator. Elsewhere, we have found that addressees can disambiguate a speaker's referring expressions earlier when they can see where the speaker is looking ([16][17]).

4. IMPLICATIONS FOR FUTURE WORK

This study demonstrates that one person's eye gaze, even when produced instrumentally (in the service of doing a task and in the absence of an intention to communicate), can provide useful cues to another person of roughly equal skill in a software debugging task. It further confirms that programming is an appropriate visuo-spatial domain for examining the effects of shared visual attention on collaboration.

Questions that remain to be answered are the extent to which eye gaze may be useful to people collaborating in real-time, the extent to which it may provide benefits beyond those of intentionally pointing with a mouse, how it may interact with having a voice channel, and whether viewing an expert programmer's eye gaze can provide interpretable cues to a novice. Novice programmers may be unable to quickly interpret experts' gazes to a slice, and so the gaze trace may indicate merely *where* a bug is without helping them with *what* it is. On the other hand, if there were a voice channel and the novice could listen to the expert explain a bug in real time, seeing the expert's looks to the relevant slice of code may be helpful beyond what is gained from seeing only a mouse cursor intentionally indicating an edit point in the code.

Even if novices lack the experience and knowledge base needed to benefit from experts' eyegaze patterns, there may exist potential benefit in using eye tracking to support collaborationñ but in the other direction: An expert who can observe a novice's eye gaze may be able to track whether the novice is understanding an explanation and to better diagnose the novice's state of (mis)understanding. As eye trackers become cheaper, less intrusive, and more accurate, we expect that eye gaze will become a key element in interfaces that support mediated communication and collaboration.

5. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grants No. 0082602 and 9980013. We thank Gregory Zelinsky and Richard Gerrig for their contributions to this work.

6. REFERENCES

[1] Ballard, D. H., Hayhoe, M. M., Li, F., and Whitehead, S. D. (1992). Hand-eye coordination during sequential tasks. *Philosophical Transactions of the Royal Society of London*, *B*(*337*), 331-339.

[2] Ballard, D. H., Hayhoe, M. M., and Pelz, J. B. (1995). Memory representations in natural tasks. *Journal of Cognitive Neuroscience*, 7(1), 66-80.

[3] Brennan, S. E. (1990). *Seeking and providing evidence for mutual understanding.* Unpublished doctoral dissertation, Stanford University.

[4] Brennan, S. E. (In press). How conversation is shaped by visual and spoken evidence. In J. Trueswell and M. Tanenhaus (Eds.), World Situated Language Use: Psycholinguistic, Linguistic and Computational Perspectives on Bridging the Product and Action Traditions. Cambridge, MA: MIT Press.

[5] Brennan, S. E. (1995). Centering attention in discourse. Language and Cognitive Processes, 10, 137-167.

[6] Brennan, S. E. and Lockridge, C. B. Monitoring an addressee's visual attention: Effects of visual co-presence on referring in conversation. Unpublished manuscript.

[7] Clark, H.H. and Brennan, S.E. (1991). Grounding in communication. In L.B. Resnick, J.M. Levine and S.D. Teasley (Eds.) Perspectives on socially shared cognition (pp. 127-149). Washington, DC: American Psychological Association.

[8] Doherty-Sneddon, G., Anderson, A., OíMalley, C., Langton, S., Garrod, S., and Bruce, V. (1997). Face-to-face and video-mediated communication: A comparison of dialogue structure and task performance. *Journal of Experimental Psychology: Applied, 3*, 105-125.

[9] Dourish, P., Adler, A., Bellotti, V., and Henderson, A. (1996). Your place or mine? Learning from long-term use of audio-video communication. *Computer Supported Cooperative Work, 5*, 33-62.

[10] Fix, V., Wiedenbeck, S., and Scholtz, J. Mental representations of programs by novices and experts. *Proceedings of the SIGCHI conference on human factors in computing systems (CHI 93)*

[11] Grice, H. P. (1975). Logic and conversation (from the William James lectures, Harvard University, 1967). In P. Cole and J. Morgan (Eds.), *Syntax and semantics 3: Speech acts*. (pp. 41-58). New York: Academic Press.

[12] Grosz, B. J., Joshi, A. K., and Weinstein, S. (1995). Centering: A framework for modelling the local coherence of discourse. Computational Linguistics.

[13] Grosz, B. J. and Sidner, C. L. (1986). Attention, intentions, and the structure of discourse. *Computational Linguistics, 12*, 175-204.

[14] Gugerty, L and Olson, G.M. (1986) ìDebugging by Skilled and Novice Programmers.î *CHI í86 Proceedings*,171-174.

[15] Gutwin, C., Greenberg, S., and Roseman, M. (1996). Supporting awareness of others in groupware. *ACM CHI í96 Conference Companion*.

[16] Hanna, J. E, and Brennan, S. E. (2003). Eye gaze has immediate effects on reference resolution in conversation. *Abstracts of the Psychonomic Society, 44th Annual Meeting* (p. 124), Vancouver, Canada.

[17] Hanna, J.E. and Brennan, S. E. (2004). Using a Speaker's Eye gaze During Comprehension: A Cue Both Rapid and Flexible. Abstract, *17th Annual CUNY Conference on Human Sentence Processing,* College Park, MD.

[18] Hindmarsh, Fraser, Heath, Benford, and Greenhalgh, 1998.Hindmarsh, J. Fraser, M., Heath, C., Benford, S., and Greenhalgh, C. (1998). Fragmented interaction: establishing mutual orientation in virtual environments. *Proceedings ACM Conference on Computer supported Cooperative Work* (pp. 217-226), November 14 - 18, 1998, Seattle, WA.

[19] Isaacs, E. A., Tang, J. C., and Morris, T. (1996). Piazza: A desktop environment supporting impromptu and planned interactions. *CSCW í96*, 315-324.

[20] Ishii, H., Kobayshi, M., and Grudin, J. (1992). Integration of inter-personal space and shared workspace: ClearBoard design and experiments. In *Proceedings, ACM Conference on Computer-Supported Cooperative Work, CSCW í92* (pp. 33-42). New York: ACM Press.

[21] Jacob, R. J. K. (1995). Eye tracking in advanced interface design. In W. Barfield and T. A. Furness (Eds.), *Virtual environments and advanced interface design* (pp. 258-308). New York: OxfordUniversity Press.

[22] Karsenty, L. (1999). Cooperative work and shared visual context: An empirical study of comprehension problems in side-by-side and remote help dialogues. *Human-Computer Interaction, 14*, 283-316.

[23] Koenemann, J., and Robertson, S. P. (1991) "Expert Problem Solving Strategies for Program Comprehension." *Human Factors in Computing Systems: CHI í91*, 125-130.

[24] Kraut, R. E., Fussell, S. R., Brennan, S. E., and Siegel, J. (2002). Understanding effects of proximity on collaboration : Implications for technologies to support remote collaborative

work. In P. Hinds and S. Kiesler, *Distributed work* (pp. 137-162). Cambridge, MA: MIT Press.

[25] Nardi, B., Schwartz, H. Kuchinsky, A., Leichner, R., Whittaker, S., and Sclabassi, R. (1993). Turning away from talking heads: An analysis of video-as-data. *Proceedings, CHI '99, Human Factors in Computing Systems*, pp. 327-334. New York: ACM.

[26] Ramalingam, V. and Wiedenbeck, S. (1997). ìAn Empirical Study of Novice Program Comprehension in the Imperative and Object-Oriented Stylesî. *Papers presented at the seventh workshop on Empirical studies of programmers*, 124-139.

[27] Sellen, A. J. (1995). Remote conversations: The effects of mediating talk with technology. *Human-Computer Interaction, 10*, 401-444.

[28] Soloway, E, Bonar, J. and Elrich, K (1982). ìTapping into tacit programming knowledgeî. *Proceedings of the first major conference on Human factors in computers systems,*.52-57.

[29] Spohrer, J.C, and Soloway, E. (1986a). ìAlternatives to construct-based programming misconceptionsî. ACM SIGCHI Bulletin , *Conference proceedings on Human factors in computing systems. Volume 17 Issue 4.*

[30] Spohrer, J. C. and Soloway, E. (1986b) ìNovice mistakes: are the folk wisdoms correct?î. *Communications of the ACM. Volume 29 Issue 7.*

[31] Spohrer, J. C., Soloway, E, and Pope, E. (1985). ìWhere the Bugs Areî. *CHI ì85Proceedings*, 47-53.

[32] Tang, J. and Isaacs, E. (1993). Why do users like video? Studies of multimedia-supported collaboration. *Computer Supported Cooperative Work, 11*, 163-196.

[33] Weiser, M. (1982) ìProgrammers Use Slices When Debuggingî. *Communications of the ACM. Volume 25 Issue 7*, 446-453.

[34] Whittaker, S. (1995) Rethinking video as a technology for interpersonal communications: theory and design implications. *International Journal of Man-Machine Studies, 42*, 501-529.

[35] Whittaker, S. and Geelhoed, E. (1993). Shared workspaces: How do they work and when are they useful? *International Journal of Man-Machine Studies, 39*, 813-842